ALGORITHMIC ASPECTS OF CONSTRAINED UNIT DISK GRAPHS

By

Heinz Breu

B.Sc. University of British Columbia, 1978

M.Sc. University of British Columbia, 1980

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

COMPUTER SCIENCE

We accept this thesis as conforming

to the required standard

. . . .

. . . .

. . . .

. . . .

. . . .

THE UNIVERSITY OF BRITISH COLUMBIA

April 1996

Computer Science
The University of British Columbia
2366 Main Mall
Vancouver, Canada
V6T 1Z4

Date:

Abstract

Computational problems on graphs often arise in two- or three-dimensional geometric contexts. Such problems include assigning channels to radio transmitters (graph colouring), physically routing traces on a printed circuit board (graph drawing), and modelling molecules. It is reasonable to expect that natural graph problems have more efficient solutions when restricted to such geometric graphs. Unfortunately, many familiar **NP**-complete problems remain **NP**-complete on geometric graphs.

Indifference graphs arise in a one-dimensional geometric context; they are the intersection graphs of unit intervals on the line. Many **NP**-complete problems on arbitrary graphs do have efficient solutions on indifference graphs. Yet these same problems remain **NP**-complete for the intersection graphs of unit disks in the plane (*unit disk graphs*), a natural two-dimensional generalization of indifference graphs. What accounts for this situation, and how can algorithms be designed to deal with it?

To study these issues, this thesis identifies a range of subclasses of unit disk graphs in which the second spatial dimension is gradually introduced. More specifically, τ-*strip graphs* "interpolate" between unit disk graphs and indifference graphs; they are the intersection graphs of unit-diameter disks whose centres are constrained to lie in a strip of thickness τ. This thesis studies algorithmic and structural aspects of varying the value τ for τ-strip graphs.

The thesis takes significant steps towards characterizing, recognizing, and laying out strip graphs. We will also see how to develop algorithms for several problems on strip graphs, and how to exploit their geometric representation. In particular, we will see that problems become especially tractable when the strips are "thin" (τ is small) or "discrete"

(the number of possible y-coordinates for the disks is small). Note again that indifference graphs are the thinnest ($\tau = 0$) and most discrete (one y-coordinate) of the nontrivial τ-strip graphs.

The immediate results of this research concern algorithms for a specific class of graphs. The real contribution of this research is the elucidation of when and where geometry can be exploited in the development of efficient graph theoretic algorithms.

Table of Contents

List of Tables

List of Figures

Acknowledgement

This thesis would not have been possible without my mentor and supervisor, Professor David G. Kirkpatrick[1]. It is not possible to say that his influence is stronger in any one chapter, since every part of this thesis has felt his touch.

My thesis committee comprised Professors Richard Anstee, Alain Fournier, Maria Klawe, Nick Pippenger, and Alan Wagner. Thanks for many comments, especially in setting early directions.

I am grateful to Professors Y. Daniel Liang, Indiana Purdue University at Fort Wayne; Lorna Stewart, University of Alberta; and C. Pandu Rangan, Indian Institute of Technology Madras for helpful discussions and references related to domination algorithms on cocomparability graphs (§4.2). I am also grateful to Professor Jan Kratochvíl, Charles University, Prague, for helpful discussions on the **NP**-hardness proof for unit disk graphs (§3.4), and Dr. Madhav Marathe, Los Alamos National Labs, for extended discussions on approximation algorithms for unit disk graphs (§3.3.3).

Thanks to Hewlett-Packard Company, and especially Hewlett-Packard Laboratories, for keeping me on leave-of-absence for so many years.

I also must thank two long-time friends for greatly enriching my years at U.B.C.: Professor David Lowe, for welcoming us to Vancouver, for always showing a keen interest in my work, for always being encouraging, and for teaching me how to teach; and Professor Trevor Hurwitz, for many lunches and stimulating discussions. Many new friends, far too many to acknowledge individually, have also influenced this thesis. Still, two have made unusual impacts: Professor Yossi Gil, the Technion, Haifa; and Dr. Will Evans.

[1]Individuals whose affiliations are not explicitly stated are with the University of British Columbia.

My parents have been an enormous influence. I would not be here without their early encouragement of my scientific career. They have also contributed greatly to the resolution of those daily tasks that face us all, yet take so much of our time.

Finally, thank you Wendy, my wife and partner for 16 years. You put your own interests on hold for over six long years so that I might pursue this research. In particular, you supported us financially when you would much rather have participated full time in the life of our son Lorenz, now four years old. To you I dedicate this thesis.

Chapter 1

Introduction

1.1 Prologue

"It just can't be this hard," mumbled Alice[1] to herself. "I wouldn't feel so bad if my graph theoretic model didn't *fit* the problem so well, but it seems so natural. It really simplifies the statement of my problem; it just doesn't seem to lead to an efficient algorithm." She found herself ruminating over the events of the last few months.

It all began with a big project meeting at Blue Sky Airlines ("The airline where the rubber meets the sky"), where Alice designs algorithms for a living. Blue Sky had equipped each airplane in its fleet with a radio beacon. Every beacon has the same range, which is uniform in all directions. Over the next few weeks, Alice discovered that one of her tasks is to solve the "midnight bell problem", as she called it. Every day at midnight (Vancouver time), every plane in a certain trans-Pacific corridor must either send a beacon signal, or receive one from at least one other plane. Blue Sky will know the location (including the altitude) of every plane in the corridor each midnight. Alice's job is to minimize the number of beacons that must send a signal.

Naturally, Alice does not plan to show up every midnight to tell Blue Sky which beacons to buzz. Her problem is really to design an *algorithm* to solve this problem for her, given the number and locations of the planes as input. Furthermore, Blue Sky expects to have more than 50 airplanes in the corridor every night by late next decade

[1]Alice and her company are fictional. This prologue, and its epilogue in the Conclusion chapter, is intended only to motivate the theoretical problems addressed in this thesis, and to strengthen your intuition.

("The sky's the limit, Alice!"), so it had better be a *good* algorithm.

Alice thought about this problem for some time, formulating suitable abstractions. Eventually she constructed a graph theoretic model of her problem. The airplanes are vertices in the graph, and two vertices are adjacent in the graph if they are within beacon range. Now, she only needs to find a minimum cardinality subset of vertices (the "transmitters") such that every other vertex ("receiver") is adjacent to a transmitter. Alice was pleased with her formulation; she likes using graph theory and knows that there are many algorithms available for many different problems. But this graph theoretical problem looks like it might be hard.

She consulted her favorite reference book on these matters, *Computers and Intractability: a Guide to the Theory of* **NP**-*Completeness*, by Garey and Johnson [GJ79]. After a few hours—Alice is easily distracted by the many interesting problems to be found in Garey and Johnson—she discovered that the corresponding decision problem is called the DOMINATING SET problem, and that it is indeed **NP**-complete.

What could she do now? Fifty airplanes are not *that* many, but still too many to consider every subset of airplanes, in exponential time. She could develop a heuristic solution, but this approach always leaves her feeling unsatisfied. Perhaps graph theory is a little *too* general. After all, her original problem had a very geometric flavour to it, involving airplanes in a shallow corridor and beacons with circular ranges. What happened to all the geometry? And how could it be used to solve her problem, anyway? She could think of no other option but to start the whole abstraction process again. Alice slumped heavily over her desk, with the dreadful feeling that she had run out of alternatives.

1.2 Geometric Constraints in Graph Theory

Let us propose an alternative to Alice: exploit geometric constraints in graph theoretic problems. We will study this advice by deriving a class of graphs from geometric situations. In particular, this thesis studies *unit disk graphs*, which are the intersection graphs of closed unit-diameter disks in the plane. Although this thesis may on occasion draw your attention to related problems set in higher dimensional space, its primary concern is with the plane, that is, with two-dimensional space. Clearly, unit disk graphs capture some two-dimensional constraints. Some subclasses of unit disk graphs capture even more geometric constraints. In particular, if we constrain the centres of the circles to be collinear, then we call the graphs *indifference graphs* or *unit interval graphs*. Clearly, indifference graphs capture some one-dimensional constraints. With some care, we can move "gradually" from unit interval graphs to unit disk graphs, as outlined below. This thesis explores algorithmic implications of this gradual movement.

As hinted in the prologue, this thesis studies the complexity of some **NP**-complete graph theoretical problems restricted to these classes of unit disk graphs. Graph theoretic problems on these classes exhibit diverse behaviour. Some problems remain **NP**-complete for some of these classes, but some problems admit polynomial time solutions, and some problems may be efficiently approximated. Sometimes there are efficient algorithms that require a model for the graph, that is, a set of disks that realize the graph. We are therefore also interested in the complexity of recognizing unit disk graphs, or of constructing a realization. Unfortunately, we will see that the decision problem corresponding to both problems is **NP**-hard. On the other hand, there are algorithms for some problems that do not require models.

A graph is a *unit disk graph* if each vertex can be mapped to a closed, unit diameter disk in the plane such that two vertices are adjacent (in the graph) if and only if their corresponding disks intersect (on the plane). Since there are other definitions for unit disk graphs, let us refer to this mapping of vertices to disks as a *unit disk model.* Another definition is that a graph is a unit disk graph if each vertex can be mapped to a point in the plane such that two vertices are adjacent (in the graph) if and only if their corresponding points are within unit distance of one another. See Section 1.3.5 for a more careful definition of these terms. Such a mapping of vertices to points is called a *proximity model* of the graph. Clearly, there is a one-to-one correspondence between these families of models: the centre of a disk is the mapped-to point. Clearly also, the unit of distance is of no great consequence, since the models of graphs under one unit can be transformed into another by scaling. This thesis uses both models for the analysis of algorithms, and sometimes also for their implementation.

Alternatively, we can think of the model as generating the unit disk graph. Construct a unit disk graph from a set of points (or unit disks) in the plane by identifying vertices with points and by putting an edge between two vertices if the distance between them is at most unit distance (or if the disks intersect). For example, Alice's Blue Sky problem can be modelled in two dimensions (i.e., assume that the airplanes lie in a common vertical plane) by letting the unit distance be the beacon range, and by centering unit-diameter disks on the airplanes.

Alice's problem is even more constrained. Let us imagine that a "trans-Pacific corridor" is an (effectively) infinite length strip (rectangle) with thickness τ, which is determined by the least and greatest allowed altitudes. We are particularly interested in the case where τ is small compared to the (unit) range of the radio beacons, say $0 \leq \tau \leq \sqrt{3}/2$. Such unit disk graphs, where the disks lie in a thin strip, are called *τ-strip graphs* or just *strip graphs* when $\tau = \sqrt{3}/2$. The reason for the number $\sqrt{3}/2$ is that every strip graph is also a cocomparability graph (Theorem 3.7). Such graphs are perfect, which is often a hint that many graph theoretic problems involving cliques, colours, and independent sets can be solved in polynomial time. In fact, we will solve Alice's problem for cocomparability graphs in Chapter 4. Note that the class of 0-strip graphs is identical to the class of indifference graphs. Note also that every unit disk graph is a τ-strip graph for some (not necessarily small) value τ.

Cocomparability graphs properly include the class of strip graphs. Is it possible to exploit the additional constraints from strips? It is, as Chapter 5 demonstrates by solving a problem related to Alice's, namely the minimum weight Steiner set problem, even more efficiently for strip graphs than for cocomparability graphs.

We also could imagine that certain "rules of the road" apply to Blue Sky's airplanes in the corridor. For example, perhaps they restrict east bound traffic to the upper boundary of the corridor (let us hope that they also restrict west bound traffic to the lower boundary). Such a unit disk graph is called a *two-level τ-strip graph*. Chapter 6 solves another problem related to Alice's, namely the minimum weight independent dominating set problem for two-level graphs.

1.3 Conventions, Background, and Notation

The following definitions for sets (§1.3.1), geometry (§1.3.3), algorithms (§1.3.4), and graphs (§1.3.2) are standard, or as nearly so as the literature allows. This thesis assumes that you are already familiar with these basic *concepts*, but presents their definitions to avoid ambiguity. Furthermore, the body of the thesis repeats some of these definitions as they are required. You may therefore want to skim lightly over this section on a first reading, to ensure that these definitions coincide with your expectations. The section on unit disk graphs (§1.3.5) is more esoteric, so you may want to examine it more closely.

1.3.1 Sets

These definitions are common to many standard textbooks on discrete mathematics or algorithms, for example [Gol80], [CLR90], [Epp95].

The terms *set* and *element* are the undefined primitives of axiomatic set theory, but the intended meaning is that a set is a collection of elements. Some sets have special symbols: $\emptyset$ is the empty set, which contains no elements, $\mathbf{Z}$ is the set of integers, $\mathbf{Z}^+$ is the set of positive integers, and $\mathbf{R}$ is the set of real numbers. The following table summarizes basic relations on sets.

Write:	and say:	if, for all elements x in some universal set:
$x \in S$	x is in S	x is an element of set S
$x \notin S$	x is not in S	x is not an element of set S
$A \subseteq B$	A is a *subset* of B	$x \in A$ implies $x \in B$
$A = B$	A equals B	$A \subseteq B$ and $B \subseteq A$
$A \subset B$	A is a *proper subset* of B	$A \subseteq B$ but $A \neq B$
$A \supseteq B$	A is a *superset* of B	$B \subseteq A$
$A \supset B$	A is a *proper superset* of B	$B \subset A$

For example, $\sqrt{2} \in \mathbf{R}$ but $\sqrt{2} \notin \mathbf{Z}$, and $\mathbf{Z} \subset \mathbf{R}$. Usually, sets will be uppercase symbols, and elements will be lowercase. This thesis is concerned primarily with finite sets, for which the elements can often be listed explicitly or implicitly inside curly braces. For example, $\{1, 2, 3, \ldots, n\}$ denotes the set of integers from 1 to n inclusive. We can also specify sets by rule, for example $\{1, 2, 3, \ldots, n\} = \{i : i \in \mathbf{Z} \text{ and } 1 \leq i \leq n\}$. For a finite set S, let $|S|$ denote its *cardinality*, in this case the number of elements in S. For example, $|\{1, 2, 3, \ldots, n\}| = n$. A *multiset* is similar to a set, except that it may have two or more elements with the same name. For example, set $\{1, 2, 3\}$ and set $\{1, 2, 2, 3\}$ are the same and have three elements, but *multiset* $\{1, 2, 2, 3\}$ has four elements, including two occurrences of element 2.

The basic set operations in the following table are especially useful. Here, A and B are sets, and U is some "universe" set that will be clear from context.

operator:	written:	equals:
union	$A \cup B$	$\{x : x \in A \text{ or } x \in B\}$
intersection	$A \cap B$	$\{x : x \in A \text{ and } x \in B\}$
difference	$A \setminus B$	$\{x : x \in A \text{ and } x \notin B\}$
complement	$\overline{A}$	$\{x : x \in U \text{ and } x \notin A\}$
Cartesian Product	$A \times B$	$\{(a, b) : a \in A \text{ and } b \in B\}$

Two sets A and B are *disjoint* if $A \cap B = \emptyset$. If the operands A and B of the union operator are disjoint, we may write the *disjoint union* $A+B$ instead of $A \cup B$ to emphasize this fact. The Cartesian product operation is also defined on more than two sets:

$$S_1 \times S_2 \times \cdots \times S_k = \{(s_1, s_2, \ldots, s_k) : s_1 \in S_1,\ s_2 \in S_2,\ \ldots,\ \text{and } s_k \in S_k\}.$$

The elements of the Cartesian product of k sets are called (ordered) *k-tuples*. When all k sets are the same, it is traditional to write $S \times S \times \cdots \times S = S^k$. For example, $\mathbf{R}^2$ denotes the real plane as a set of *coordinates* $(x, y) \in \mathbf{R}^2$.

A *k-ary relation* on a Cartesian product of sets $S_1 \times S_2 \times \cdots \times S_k$ is just a subset of this product. This thesis is mainly concerned with *binary relations*, for which $k = 2$. More specifically, it is primarily concerned with binary relations on a Cartesian product of the same set. Therefore, say that a *(binary) relation on a set S* is a pair (S, R), where $R \subseteq S^2$. Sometimes we will write aRb to mean $(a, b) \in R$. There are a few standard properties on binary relations, as summarized by the following table.

A binary relation (S, R) is said to be:	if, for all $a, b, c \in S$:
reflexive	$(a, a) \in R$
irreflexive	$(a, a) \notin R$
symmetric	$(a, b) \in R$ implies $(b, a) \in R$
asymmetric	$(a, b) \in R$ implies $(b, a) \notin R$
antisymmetric	$(a, b) \in R$ and $(b, a) \in R$ implies $a = b$
transitive	$(a, b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$
complete	$a \neq b$ implies $(a, b) \in R$ or $(b, a) \in R$
strongly complete	$(a, b) \in R$ or $(b, a) \in R$

It is sometimes advantageous to force a relation to have one or more of these properties. The following table defines a set of operations on relations designed for this purpose.

This operation on relation R	yields the relation:
inverse	$R^{-1} = \{(a, b) : (b, a) \in R\}$
reflexive closure	$R \cup \{(a, a) : (a, a) \in R\}$
symmetric closure	$R \cup \{(b, a) : (a, b) \in R\} = R \cup R^{-1}$
transitive closure	the smallest transitive superset of R
transitive reduction	the smallest relation having the same transitive closure as R

Some relations are given special names, depending on the standard properties they satisfy. The following table summarizes the relations used in this thesis.

A Binary Relation (S, R) is said to be a:	if it is:
partial order	reflexive, antisymmetric, and transitive
strict partial order	irreflexive, asymmetric, and transitive
linear order	a strongly complete, partial order
strict linear order	a complete, strict partial order
equivalence relation	reflexive, symmetric, and transitive (If $(S, \equiv)$ is an equivalence relation, and $a \in S$, then the *equivalence class* of a is the set $[a] = \{x : x \equiv a\}$.)

Note that the literature also refers to linear orders as *total orders*, *complete orders*, and *simple orders*. A strict linear order can also be specified by listing its elements $(a_1, a_2, \ldots, a_n)$ with the understanding that $a_i < a_j$ if and only if $i < j$. This thesis is mainly concerned with strict partial orders (and strict linear orders), which are sometimes written as pairs $(S, <)$. Furthermore, to distinguish the relation from the set on which it is defined, we will sometimes want to refer to the relation R as a partial order, and the pair (S, R) as a *partially ordered set*, or more economically, as a *poset*. An element a in a poset $(S, <)$ is *maximal* (respectively *minimal*) if $a < x$ (respectively $x < a$) does not hold for any element $x \in S$. In particular, a set S in a family of sets $\mathcal{F}$ is *maximal* if it is a maximal element in the poset $(\mathcal{F}, \subset)$. The *restriction* (S', R') of a relation (S, R) to a subset $S' \subseteq S$ is defined by the equation $R' = \{(a, b) : a, b \in S' \text{ and } (a, b) \in R\}$. Note that the restriction of a poset is also a poset. A *linear extension* of a strict partial order R is a complete strict partial order R', where $R \subseteq R'$. It is perhaps not obvious that such a linear extension exists; you can always generate one by listing (topologically sorting, [CLR90] pages 485–488) its elements as follows. Remove a maximal element from S, recursively list the partial order restricted to $S \setminus \{m\}$, and append m to the end of the list.

The operators union, intersection, and difference are defined also for binary relations on sets, by operating on the set and the relation. For example, the *intersection* $(S_1, R_1) \cap (S_2, R_2)$ of two binary relations is the relation $(S_1 \cap S_2, R_1 \cap R_2)$. In nearly all cases, this thesis applies set operators only to binary relations on the same set.

The *dimension* of a partial order P is the minimum number of linear orders whose intersection is P. Note that, if P is the intersection of k linear orders, then each linear order is a linear extension of P. Note also, that P is the intersection of all of its linear extensions, so the notion of dimension is well-defined.

An *interval* is a contiguous subset of a linearly ordered set. There are *open, closed,* and *partially closed* intervals defined by the following table.

open	$(a, b) = \{x : a < x < b\}$
closed	$[a, b] = \{x : a \leq x \leq b\}$
partially closed	$(a, b] = \{x : a < x \leq b\}$
partially closed	$[a, b) = \{x : a \leq x < b\}$

A poset (S, R) is called an *interval order* if S is a set of intervals, and

$$R = \{((a, b), (c, d)) : (a, b) \in S,\ (c, d) \in S, \text{ and } b < c\}.$$

The *interval order dimension* of a poset P is the smallest number of interval orders whose intersection is P.

1.3.2 Graphs

For the most part, the graph theoretic definitions in this thesis conform to those in Golumbic's book [Gol80] on algorithmic graph theory. A *graph* $G = (V, E)$ is a set of vertices V and an irreflexive binary relation E, called *edges*, on the vertices. We also write $V(G) = V$ and $E(G) = E$. Unless mentioned otherwise, both sets are finite; the *order of a graph* G is the value $|V(G)|$. A *multigraph* is similar to a graph, except that E is a multiset. If, in addition, E is not irreflexive, then G is called a *pseudograph.*

If E is symmetric, the graph is said to be *undirected.* In this case, it is sometimes convenient to assume that edges (a, b) and (b, a) are the same edge, which is also said to be *undirected.* Since graphs are irreflexive, the *complement* $\overline{G}$ *of a graph* G is defined differently than the complement of a relation, it is the graph $\overline{G} = (V, \overline{E})$ where $\overline{E} = \{(u, v) : (u, v) \notin E \text{ and } u \neq v\}$. If E is not symmetric, then G is called a *directed graph* and has *directed edges* or *arcs.* We will often write A instead of E for arcs. An *oriented graph* is a directed graph $G = (V, A)$ where A is asymmetric. An *orientation* of an undirected graph $G = (V, E)$ is an oriented subgraph $H = (V, A)$ where $E = A + A^{-1}$ (the arc set A is also called an *orientation* of the edge set E).

The adjectives *dense* and *sparse* are used informally in this thesis. Roughly, a graph G is *dense* if $|E(G)| = \Theta(V^2)$ and *sparse* otherwise.

Two vertices u and v in V are said to be *adjacent* if $(u, v) \in E$; vertex u is said to be *adjacent from* vertex v, and vertex v is said to be *adjacent to* vertex u. Two adjacent vertices are said to be *connected by an edge.* If (u, v) is an edge, then vertices u and v are said to be its *endpoints.* An edge is said to be *incident* to its endpoints, and two edges are *adjacent* if they have a common endpoint. Graph vertices will usually be drawn as small circles. An undirected edge will be drawn as a solid, usually straight, line segment between the circles corresponding to its endpoints. A directed edge (u, v) will be drawn as an arrow from circle u to circle v. When an edge in the complement of a graph needs to be emphasized, it will be drawn as a dotted line segment. The following table summarizes some familiar parameters of vertices in graphs.

Name	Symbol	Meaning
adjacency list	$\mathrm{Adj}(v)$	$\{u : (v, u) \in E\}$
out-neighbourhood	$\mathrm{N}^{+}(v)$	$\mathrm{Adj}(v) \cup \{v\}$
in-neighbourhood	$\mathrm{N}^{-}(v)$	$\{u : (u, v) \in E\} \cup \{v\}$
neighbourhood (undirected)	$\mathrm{N}(v)$	$\mathrm{N}^{+}(v)$
indegree	$\mathrm{indegree}(v)$	$\lvert\{(u, v) : (u, v) \in E\}\rvert$
outdegree	$\mathrm{outdegree}(v)$	$\lvert\{(v, u) : (v, u) \in E\}\rvert$
degree (for undirected graphs)	$\deg(v)$	$\mathrm{outdegree}(v)$

Note that the adjacency list of v is just the set of vertices adjacent to v, and that $\deg(v) = \mathrm{indegree}(v) = \mathrm{outdegree}(v) = |\mathrm{Adj}(v)|$ for undirected graphs. The neighbourhood $\mathrm{N}(S)$ of a subset $S \subseteq V(G)$ is the union of the neighbourhoods[2] of its elements. That is, $\mathrm{N}(S) = \{u : u \in N(v) \text{ for some } v \in S\}$. A *source* in a directed graph G is a minimal vertex in G (that is, one with indegree 0). Similarly, a *sink* in a directed graph G is a maximal vertex in G (that is, one with outdegree 0).

A graph $G_s = (V_s, E_s)$ is a *subgraph* of a graph $G = (V, E)$, written $G_s \subseteq G$, if $V_s \subseteq V$ and $E_s \subseteq E$. Subgraph G_s is *spanning* if $V_s = V$. The subgraph of $G = (V, E)$ *induced by* (*induced on, generated by*) a subset of vertices $U \subseteq V$ is the graph $G(U) = (U, E_U)$ where E_U is the set of all edges in E that have both endpoints in U. A class of graphs is *hereditary* or *closed under taking induced subgraphs* if every induced subgraph of every graph in the class is also a graph in the class. When this is the case (and only when this is case), it makes sense to consider a *forbidden subgraph characterization* of the class. A graph F is a *forbidden subgraph* of the class if it is not in the class, but every induced subgraph of F is in the class. The *forbidden subgraph characterization* of a class of hereditary graphs is the set of all forbidden subgraphs.

[2]The neighbourhood of a vertex is also sometimes called the *closed neighbourhood* of the vertex. This is because some authors use "neighbourhood" to mean the adjacency list.

The *complement* $\overline{G}$ of a graph $G = (V, E)$ is the graph $\overline{G} = (V, \overline{E})$. A subset of vertices $U \subseteq V$ is *independent* if no pair of vertices in U is connected by an edge, that is, if $E(G(U)) = \emptyset$. A subgraph K of a graph G is a *clique* (or *completely connected*) if every pair of vertices in K is connected by an edge, that is, if $E(\overline{K}) = \emptyset$. A *matching* in a graph G is a subset of edges $M \subseteq E(G)$ such that no two edges are adjacent.

Let D be a subset of the vertices V of a graph $G = (V, E)$. Subset D is said to *dominate* a vertex $v \in V$ if $v \in D$ or if v is adjacent to some vertex in D. Subset D is a *dominating set*, and said to dominate the graph G (or the vertices V), if D dominates every vertex in V. Subset D is said to be a *connected dominating set* if it is dominating and the subgraph it induces is connected. Subset D is an *independent dominating set* if it is dominating and the subgraph it induces has no edges. Subset D is a *total dominating set* if every vertex in V (including those in D) is adjacent to some vertex in D. Given a graph and a required set $R \subseteq V$, a subset $S \subseteq V \setminus R$ is a *Steiner set* if the subgraph induced by $R \cup S$ is connected.

In general, a *maximum subgraph* satisfying some specified property is one that attains the greatest cardinality. For example, a *maximum clique* is a clique that has at least as many vertices as any other clique, and a *maximum matching* is one that has at least as many edges as any other matching.

A graph $G = (V, E, <)$ is *partially ordered* if $(V, <)$ is a partial order on the vertex set. Often this partial order will be linear. If G_1 and G_2 are subgraphs in a linearly-ordered graph, then $G_1 < G_2$ if $u < v$ for all $u \in V(G_1)$ and $v \in V(G_2)$. The subgraphs are said to *overlap in the order* if neither $G_1 < G_2$ nor $G_2 < G_1$.

The vertices V of a graph G are *weighted* by a function $w : V \rightarrow \mathbf{R}$. Similarly, the edges E of a graph G are *weighted* by a function $w : E \rightarrow \mathbf{R}$. A *weighted graph* $G = (V, E, w)$ may be either a vertex weighted graph or an edge weighted graph, depending on context. The *weight of a graph* is typically the sum of the weights of its vertices (or edges), but may be redefined to be some other function of its constituent weights.

A *chain of length* k $(v_0, v_1, v_2, \ldots, v_k)$ in a graph G is a sequence of vertices from $V(G)$ such that $(v_{i-1}, v_i) \in E$ or $(v_i, v_{i-1}) \in E$ for $i = 1, 2, \ldots, k$. The vertices v_0 and v_k are called the *endpoints* of the chain, and the chain is *between* its endpoints. A graph is *connected* (and a digraph is *weakly connected*) if there is a chain between every pair of vertices. A *path of length* k $(v_0, v_1, v_2, \ldots, v_k)$ in a graph G is a sequence of $V(G)$ such that $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, k$. A directed graph is *strongly connected* if there is a path between every pair of vertices. A *cycle of length* k is a path $(v_0, v_1, \ldots, v_k = v_0)$. Note that all cycles are paths, and all paths are chains. A chain or path of length k is called *simple* if $i \neq j$ implies $v_i \neq v_j$ for all $i, j \leq k$. A cycle of length k is called *simple* if $i \neq j$ implies $v_i \neq v_j$ for all $i, j < k$. It is sometimes convenient to treat a chain simply as a *set* of vertices or edges by context. Let $P = (v_0, v_1, \ldots, v_k)$ be a chain. An edge $(a, b) \in E$ is an *edge of* P if $(a, b) = (v_{i-1}, v_i)$ or $(a, b) = (v_i, v_{i-1})$ for some $i \in [1, k]$. An edge $(a, b) \in E$ is a *chord of* P if $a \in P$ and $b \in P$, but $(a, b) \notin P$ and $(b, a) \notin P$. A chain (or path or cycle) is *chordless* if it is simple and has no chords.

Some graphs prove to be so useful (and ubiquitous) that they have been given their own names, as the following table illustrates.

Name	Symbol	Meaning
complete graph on n vertices	K_n	$G = (V, E)$ where $E = \{(u, v) : u, v \in V\}$
complete bipartite graph on $m + n$ vertices	$K_{m,n}$	$G = (U, V, E)$ where $E = \{(u, v) : u \in U$ and $v \in V\}$
chordless cycle on n vertices	C_n	$G = (\{1, 2, \ldots, n\}, E)$ where $E = \{(i, (i \bmod n) + 1) : i \in [1, n]\}$
star on $n + 1$ vertices	$K_{1,n}$	
square	C_4	
claw	$K_{1,3}$	
triangle	$K_3 = C_3$	

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be *isomorphic* if there is a bijection (an *isomorphism*) $f : V_1 \to V_2$ that preserves adjacency, that is, $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$ for all $u, v \in V_1$. Clearly, the relation of being isomorphic, also called *isomorphism*, is an equivalence relation, that is, it is relexive, symmetric, and transitive.

The following table defines some well-known graph parameters studied in this thesis. Note that the value of any one of these parameters would be the same for all graphs in an isomorphism equivalence class. Therefore these parameters are also called *graph invariants* (they are invariant up to isomorphism).

Name	Symbol	Meaning
(maximum) graph degree	$\Delta(G)$	$\max\{\deg(v) : v \in V(G)\}$
minimum graph degree	$\delta(G)$	$\min\{\deg(v) : v \in V(G)\}$
(maximum) graph indegree	$\Delta_{in}(G)$	$\max\{\text{indegree}(v) : v \in V(G)\}$
minimum graph indegree	$\delta_{in}(G)$	$\min\{\text{indegree}(v) : v \in V(G)\}$
independence (or stability) number	$\alpha(G)$	cardinality of a maximum independent set in G
clique number	$\omega(G)$	cardinality of a maximum clique in G
clique cover number	$k(G)$	least number of cliques for which every $v \in V$ is in some clique.
chromatic number	$\chi(G)$	least number of colours to colour G. (A graph $G = (V, E)$ is *(correctly) coloured* by a function $c : V \to \mathbf{Z}^+$ if $(u, v) \in E$ implies $c(u) \neq c(v)$.)

A graph is said to be *k-partite* if its vertex set can be partitioned into k non-empty independent sets, that is, if its chromatic number is at most k. In this thesis, the most important value for k is 2; such graphs are called *bipartite*. To emphasize a bipartition of a graph's vertex set, it is traditional to write $G = (U, V, E)$, where vertex sets U and V are both independent sets. A graph is said to be *cobipartite* if its complement is bipartite.

A *planar graph* is one that can be drawn on the plane such that edges may share a vertex, but do not otherwise cross. Such a drawing is called a *plane graph*. Kuratowski characterized planar graphs in 1930. We will need the following definitions to appreciate his theorem. To *subdivide* an edge (u, v) in a graph G, replace it with a path $(u, u_1, u_2, \ldots, u_k = v)$ of length $k \geq 1$. A graph H is a *subdivision* of a graph G if H

can be constructed from G by subdividing some of its edges. Finally, two graphs are *homeomorphic* if they are subdivisions of the same graph.

Theorem 1.1 (Kuratowski [Kur30]) *Every non-planar graph has a subgraph homeomorphic to either K_5 or $K_{3,3}$.*

The binary set (and binary relation) operators are defined for graphs also, by operating on the edge and vertex sets. For example, the intersection of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is $G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2)$. In nearly all cases, this thesis will only apply set operators to graphs that have the same vertex set.

The intersection graph $\Omega(O)$ generated by a family (a multiset) O of sets, also called *the intersection graph of O*, is the graph G where $V(G) = O$ and $G(E) = \{(u, v) : u, v \in O$ and $u \cap v \neq \emptyset\}$. The *intersection graph class of a set S* is the set of all intersection graphs generated by finite (multi) subsets O of S (cf. [Sch85]). A graph G is *an S-intersection graph* if it is isomorphic to $\Omega(O)$ for some subset $O \subseteq S$. Extending the codomain of the isomorphism $f : V(G) \to O$ to the entire set S leads to the following equivalent definition. A graph G is an *S-intersection graph* if there is a mapping $f : V(G) \to S$ such that $(u, v) \in E(G)$ if and only if $f(u) \cap f(v) \neq \emptyset$. The function $f : V(G) \to S$ is called an *S-realization* of G, or a realization of G in terms of S.

1.3.3 Geometry

Points in space $\mathbf{R}^d$ (or E^d to emphasize Euclidean metric space) are denoted with lower case letters (for example, p and q) or, when there is a one-to-one correspondence between points and the vertices of a graph, just with the names of the vertices. The coordinates of a point are denoted $p = (p_1, p_2, \ldots, p_d)$. We need only a few standard operations on points (as vectors): $p + q = (p_1 + q_1, p_2 + q_2, \ldots, p_d + q_d)$, $cp = (cp_1, cp_2, \ldots, cp_n)$, where $c \in \mathbf{R}$, and $p - q = p + -1(q)$. This thesis focuses on two-dimensional space, where the coordinates of a point are denoted $p = (x_p, y_p)$.

This thesis uses three metrics in the plane: the L_1 metric, also called the *city block* (or *taxi cab* or *Manhattan*) metric; the L_2 metric, also called the *Euclidean* metric; and the L_∞ metric, also called the *maximum coordinate* (or *chess board* or *queen's move*) metric. These metrics are defined by their associated distance functions, L_k being defined by the distance function $\|\cdot\|_k : \mathbf{R}^d \to \mathbf{R}$, where

$$\|p\|_k = \left(\sum_{i=1}^{d} |p_i|^k\right)^{1/k}$$

for all $p \in \mathbf{R}^d$. The distance function for L_∞ is given by:

$$\|p\|_\infty = \lim_{k\to\infty} \|p\|_k = \max\{|p_i| : 1 \leq i \leq d\}.$$

The distance between two points p and q is given by the expression

$$\|p - q\|_k.$$

This thesis concerns itself primarily with the Euclidean metric in two dimensions. The distance function $\|p\|$ therefore will usually denote

$$\|p\|_2 = \sqrt{x_p^2 + y_p^2}.$$

If p and q are points in space, then *the (open) line segment between them* is the set $(p, q) = \{r : (1-\lambda)p + \lambda q \text{ where } \lambda \in (0,1)\}$. The *diameter* of a set of points P in space is

$$\max\{\|p - q\| : p, q \in P\}.$$

Two points in a set are said to be *diametral* if the distance between them is the diameter of the set. The *sphere of radius r about a point c* in d-dimensional space is the set

$$\{p : \|p - c\| = r \text{ and } p \in E^d\}.$$

The two-dimensional sphere is called a *circle*. The *(closed) ball of radius r about a point c* in d-dimensional space is the set

$$\{p : \|p - c\| \leq r \text{ and } p \in E^d\}.$$

The two-dimensional ball is called a *disk*. A *unit sphere, circle, ball, or disk* is a sphere, circle, ball, or disk respectively with radius 1/2, that is, with unit diameter. The *lune through a pair of points p and q* is the set

$$\{s : \|s - p\| \leq \|p - q\| \text{ and } \|s - q\| \leq \|p - q\|\}.$$

1.3.4 Algorithms

Most of the algorithms described in this thesis manipulate graphs. A traditional model of computation for such work is the Random Access Machine [AHU74], or RAM for short. Briefly, a RAM has three devices (an input, an output, and a memory), as well as a program, of course. A RAM can read from input, write to output, or store in memory, an arbitrary integer, all at unit cost in space and time. It also can execute the usual [AHU74] arithmetic and comparison operations between two integers, and program control instructions, in unit time each. Finally, this thesis uses a RAM augmented with a unit-time floor function.

Geometric computations on a RAM must restrict any input points to integer coordinates (or rational coordinates represented as pairs of integers). We can deal with the possibly irrational Euclidean distance between points by computing only the squared Euclidean distance, which is an integer (or a rational). This model, or one easily transformed to it, also will be assumed for **NP**-hardness proofs. A disadvantage of using an integer RAM is the temptation to exploit the algorithmic properties of small integers. For example, we may be tempted to sort coordinates in linear time, using counting sort [CLR90], for example. While such an exploitation can have practical benefits, particularly when the application domain is naturally discrete, it is peripheral to this investigation.

The purpose of this thesis is to investigate the role of geometric constraints in graph theory. That is, we wish to exploit geometric constraints (e.g., position, proximity, and distance) in dealing with combinatorial problems. In any representation of the underlying entities (space, points, and graphs) there are issues that are peripheral to this cause. For example, (physical) digital computers represent numbers to some finite precision. Therefore, programs written for such machines must deal with issues such as inconsistent roundoff and the need for higher precision during intermediate calculations. Where such issues are secondary to a more primary interest, as they are in this thesis, it has become customary to adopt a model of computation called a *real RAM* [PS85]. A real RAM can read, write, and store a real number, to infinite precision, in unit time and space. It also can execute the usual arithmetic and comparison operations between two real numbers in unit time each. Again, this thesis augments the real RAM with a unit-time floor function. A disadvantage of using a real RAM is the temptation to exploit the unit memory and operation times for infinite precision numbers. Again, such exploitation is not the intention of this thesis. We will use the real RAM model to describe geometric algorithms when it simplifies the presentation.

Although the RAM or real RAM is the underlying computational model, this thesis does not actually describe algorithms in terms of these primitive models. Rather, it describes algorithms with a much higher-level *pseudocode*. This pseudocode follows the conventions laid out by Cormen, Leiserson, and Rivest ([CLR90] pp. 4–5). You should have no trouble reading it even without consulting [CLR90]. The only exception is that this thesis uses the following notation for comments:

/* This is a comment. */

The thesis assumes that you are familiar with the elementary algorithms and data structures usually considered in any introductory textbook [Sed83, CLR90] on that subject. In particular, you should understand linked lists, balanced trees, and binary search. You should be familiar with the representation of a graph $G = (V, E)$ as an *adjacency list*, for example as an array of linked lists each representing $\mathrm{Adj}(v)$ for all $v \in V$, and as an *adjacency matrix*, for example as a $|V| \times |V|$ array M, where $M_{u,v} = 1$ if and only if $(u, v) \in E$. Also, you should be aware of fundamental algorithms on these graph representations, such as depth-first and breadth-first search.

As a matter of routine "programming" style, this thesis will use a *sentinel* [Sed83], in place of an explicit test for boundary conditions, wherever possible. A *sentinel* is just a "dummy" element in a data structure that holds the values for the boundary condition. For example, to search a list for a value without having to test for an end-of-list condition, first append a sentinel holding the desired value to the end of the list.

In analyzing algorithms in this thesis, we are interested primarily in their consumption of time and space resources. The *run-time of an algorithm on an input* is the number of primitive (unit-time) operations executed by the algorithm. We are interested in how the run-time changes as a function of the input size, which depends on the problem, but will always be explicit in this thesis. In nearly all cases, we will be interested in an algorithm's worst case behaviour (taken over all possible inputs) in the limit, that is, as the input size goes to infinity. This is normally expressed in *asymptotic notation*, as summarized by the following equations.

$$
\begin{aligned}
O(g(n)) &= \{f(n) : \text{there exist positive constants } c \text{ and } N \text{ such that} \\
&\qquad |f(n)| \leq c|g(n)| \text{ for all } n \geq N\} \\
\Omega(g(n)) &= \{f(n) : \text{there exist positive constants } c \text{ and } N \text{ such that} \\
&\qquad |f(n)| \geq c|g(n)| \text{ for all } n \geq N\}
\end{aligned}
$$

$$\begin{aligned}
\Theta(g(n)) &= \{f(n) : f(n) \in O(g(n)) \text{ and } g(n) \in O(f(n))\} \\
o(g(n)) &= \{f(n) : \text{for all positive constants } c \text{, there exists a positive constant } N \\
&\qquad \text{such that } |f(n)| \leq c|g(n)| \text{ for all } n \geq N\}
\end{aligned}$$

Following the lead of Cormen, Leiserson, and Rivest [CLR90], this thesis drops the set-cardinality symbols from asymptotic notation. For example, interpret $O(V \log V + E)$ as $O(|V| \log |V| + |E|)$. By convention, we write $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$, and say that $f(n)$ *is order (at most)* $g(n)$. The *computational complexity* of an algorithm refers to its worst case run-time (and memory usage). Say that an algorithm has *polynomial time (or space) complexity* if its worst case run-time (or memory usage) satisfies $f(n) = O(n^k)$ for some constant k.

Several algorithms in this thesis use matrix multiplication. The best upper bound known for the time to multiply two $n \times n$ matrices is $n^{2.376}$ [CW87]. Most often, we will be multiplying two $|V| \times |V|$ adjacency matrices, so let us abbreviate the upper time bound on this operation by $O(M(V)) = O(V^{2.376})$.

This thesis addresses several combinatorial optimization problems [GJ79, PS82]. Essentially, an *instance of a combinatorial optimization problem* is a pair (S, c), where S is a finite (hence combinatorial) set of *candidate* (or *feasible*) *solutions*, and $c : S \to \mathbf{R}$ is a cost function, telling us how expensive a feasible solution is. The goal is to find an *optimal solution* for (S, c), that is, a candidate solution s^* of minimum (or possibly maximum) cost:

$$c(s^*) = \min\{c(s) : s \in S\}.$$

A *combinatorial optimization problem* is a set of instances of a combinatorial optimization problem. For example, the *chromatic number (combinatorial optimization) problem* is to use the minimum number of positive integers to colour the vertices of a graph such that adjacent vertices do not get the same colour. An instance (S, c) of this problem comprises all (correct) ways S of assigning colours $\{1, 2, \ldots, |V(G)|\}$ to the vertices of a given graph G, and the number c of colours used for each way.

Let Π be an optimization problem. An *approximation algorithm for* Π is an algorithm that takes instances of Π as input and returns candidate solutions as output. An approximation algorithm $\mathcal{A}$ achieves a *performance ratio* R if $c(\mathcal{A}(I)) \leq R \cdot c(OPT(I))$ for every instance $I \in \Pi$, where $OPT(I)$ is an optimal solution for instance I.

Other problems in this thesis are *decision problems*: they have a yes or no answer. A decision problem corresponds to "recognizing" the yes instances of the problem. By convention, a problem name typeset in all capital letters is a decision problem, sometimes even the decision version of an optimization problem. For example, K-COLOURABILITY is the decision version, "Can the vertex set of the input graph be correctly coloured with k colours?", of the chromatic number problem.

Although researchers have identified many complexity classes, this thesis is concerned with only some of these: **P**, **NP**, **NP**-complete, and **NP**-hard. The class **P** consists of those decision problems that can be recognized (answered) in polynomial time. The class **NP** comprises those decision problems that can be "certified" in polynomial time. That is, a (decision) problem is in **NP** if there is a (necessarily polynomial size) "certificate" that the answer is yes[3], and if the certificate can be checked in polynomial time. For example, K-COLOURABILITY is in **NP** since a graph coloured with k colours can be checked in polynomial time by examining every adjacent pair of vertices in the graph. Clearly, $\mathbf{P} \subseteq \mathbf{NP}$. If you could solve every **NP**-problem in polynomial time by solving problem $\mathcal{A}$ in polynomial time, then problem $\mathcal{A}$ is said to be **NP**-*hard*.

[3] A problem is in **co-NP** if there is a polynomial-time-checkable "certificate" that the answer is no.

A decision problem is **NP**-*complete* if it is both in **NP** and **NP**-hard. The **NP**-hardness of a problem $\mathcal{A}$ is typically demonstrated by providing a polynomial-time "reduction" of some known **NP**-complete problem to problem $\mathcal{A}$. These complexity classes are normally defined more rigorously using notions of language recognition on deterministic and nondeterministic Turing machines. See the book by Garey and Johnson [GJ79] for these definitions. The significance of these definitions rests in the fact that there are no known polynomial-time algorithms for any **NP**-complete problems. That is, no one has shown that $\mathbf{P} = \mathbf{NP}$. Therefore, proving a problem to be **NP**-hard gives an effective lower bound on the complexity of any algorithm for solving the problem. On the other hand, no one has shown that $\mathbf{P} \neq \mathbf{NP}$.

1.3.5 Unit Disk, Strip, and 2-Level Graphs

The background of the last few sections allows us to define unit disk graphs more carefully. A *unit disk graph* is an S-intersection graph where S is the set of unit diameter disks in the plane. That is, a graph is a *unit disk graph* if each vertex can be mapped to a closed, unit diameter disk in the plane such that two vertices are adjacent (in the graph) if and only if their corresponding disks intersect (on the plane). This thesis briefly (§3.4) studies *disk graphs* also; these are S-intersection graphs where S is the set of disks with arbitrary (not just unit) diameter in the plane.

As mentioned in Section 1.2, an alternative to the unit disk model is the proximity model. That is, a graph is a unit disk graph if each vertex can be mapped to a point in the plane such that two vertices are adjacent (in the graph) if and only if their corresponding points are within unit distance of one another.

When a set of points P is given and generates a unit disk graph, then clearly the unit of distance *does* matter, since different graphs will be generated by different "threshold" values. When this is important, we will write $G_\delta(P) = (P, E)$ to signify the unit disk graph generated by points P, where adjacent endpoints are at most distance δ apart. The following, more formal, definition summarizes this discussion.

Definition 1.2 *The unit disk graph $G_\delta(P)$ generated by a set of points P and a real threshold δ* is the graph (P, E) where $E = \{(p,q) : \|p - q\| \leq \delta\}$. Clearly $G_\delta(P)$ is isomorphic to $G_1(\frac{1}{\delta}P)$ for every set of points P. A graph G is *a unit disk graph* if it is isomorphic to $G_1(P)$ for some $P \subset \mathbf{R}^2$. Extending the codomain of the isomorphism $f : V(G) \to P$ to the plane leads to the following equivalent definition. A graph G is a *unit disk graph* if there is a mapping $f : V(G) \to \mathbf{R}^2$ such that $(u,v) \in E(G)$ if and only if $\|f(u) - f(v)\| \leq 1$. The function f is called a *realization* of the unit disk graph. We write $f(v) = (x_f(v), y_f(v))$, where $v \in V$ and $f(S) = \{f(v) : v \in S\}$. Sometimes, when it is clear what the mapping is, we will drop the subscripts and write $x(v)$ and $y(v)$. ∎

A unit disk graph is a *strip graph* if there is a realization that maps all vertices onto a thin strip. For technical reasons that will be clear later (e.g., Theorem 3.7), we are interested only in sufficiently thin (at most $\sqrt{3}/2$ units for the Euclidean metric) strips. The following, more formal, definition summarizes.

Definition 1.3 A graph $G = (V, E)$ is a *τ-strip graph* if there is a mapping $f : V \to \mathbf{R} \times [0, \tau]$ such that $(u,v) \in E$ if and only if $\|f(u) - f(v)\| \leq 1$. The function f is called a *τ-strip realization* of the graph. Normally, we are concerned with the L_2 metric, where $\|(x,y)\| = \|(x,y)\|_2 = \sqrt{x^2 + y^2}$. For these usual L_2 τ-strip graphs, restrict τ to the interval $[0, \sqrt{3}/2]$. Occasionally, we will be interested in the L_1 metric also, where $\|(x,y)\|_1 = |x| + |y|$. For L_1 τ-strip graphs, restrict τ to the interval $[0, 1/2]$. ∎

Say that a strip graph is a *two-level graph* if it has a realization that maps every vertex onto the boundary of the strip. The following definition states this more formally.

Definition 1.4 A graph $G = (V, E)$ is a *two-level τ-strip graph* if there is a mapping $f : V \to \mathbf{R} \times \{0, \tau\}$ such that $(u, v) \in E$ if and only if $\|f(u) - f(v)\| \leq 1$. The function f is called a *two-level τ-strip realization* of the graph. ▮

1.4 Overview: A Reader's Guide

The following chapters present and derive the results discussed in the introduction. Chapter 2, Related Research, sets the stage by reviewing the context of the research reported in this thesis. In particular it defines several related classes of graphs, some of which appear in this introduction without definition. One class is particularly relevant—indifference graphs are precisely 0-strip graphs—so Chapter 2 gives it extra attention.

The remaining chapters study the "gradual introduction" of the second dimension in reverse order. They begin with Chapter 3, Unit Disk Graphs, in which the y-coordinate of the disk centres is unrestricted, as it is for the x-dimension. Chapter 3 shows how to exploit an explicit realization of a unit disk graph to find least vertex-weight paths. To do so, it makes use of a data structure whose details have been relegated to Appendix A. It also shows how to use a realization to report an adjacent vertex, and to delete a vertex, in $O(\log V)$ amortized time. This result is used by Chapters 5 and 6 to design efficient domination algorithms for strip graphs and two-level graphs. Chapter 3 continues by showing how to build the entire adjacency structure of a unit disk graph in $O(V \log V + E)$ time, given only the $|V|$ points that form the image of a realization $f : V \to \mathbf{R}^2$. Chapter 3 also shows how to prove that some problems are **NP**-complete for unit disk graphs. In particular, it conjectures that any problem that is **NP**-complete for degree 4 planar graphs is also **NP**-complete for unit disk graphs. In particular, it applies a generic transformation schema to show the **NP**-completeness of both INDEPENDENT SET and CHROMATIC NUMBER. On the other hand, it shows how to find a maximum clique in polynomial time. The main result of this chapter is that unit disk graph recognition is **NP**-hard. In proving this result, we will see that penny[4] graph, bounded-diameter-ratio coin graph, and bounded-diameter-ratio disk graph recognition are all **NP**-hard.

The next three chapters limit the second dimension to a strip of thickness τ. Since τ-strip graphs are cocomparability graphs for all $\tau \in [0, \sqrt{3}/2]$, Chapter 4 studies cocomparability graphs. In particular, it develops efficient algorithms for several dominating set problems on cocomparability graphs, which are then automatically applicable to strip graphs. In the process, the chapter develops a novel, efficient algorithm for finding a minimum weight maximal clique in a comparability graph, and therefore also a minimum weight maximal independent set in a cocomparability graph. Chapter 4 also discusses properties of the "forcing" relation on cocomparability graphs, which are used in the next chapter.

Chapter 5 studies strip graphs. It begins by demonstrating how to combine an algorithm for finding least-weight paths in unit disk graphs and an algorithm for finding least-weight Steiner sets in cocomparability graphs. The resulting algorithm for finding a minimum weight Steiner set in a strip graph is more efficient than any known for either unit disk graphs or cocomparability graphs. Chapter 5 then characterizes strip graphs with systems of difference constraints. It uses this characterization to find efficiently a realization for a strip graph that has been "levelled" and whose complement graph has been oriented to meet certain constraints. Chapter 5 continues to explore this characterization by determining which stars are τ-strip graphs for different values of τ. It also shows that the characterization implies that a strip graph is an indifference graph if it is free of induced squares and claws. The chapter concludes by characterizing those strip graphs that are trees.

[4]A graph is a *coin graph* if it is the intersection graph of a set of interior-disjoint disks. A coin graph is a *penny graph* if it is the intersection graph of a set of unit-diameter interior-disjoint disks.

Chapter 6, Two-Level Graphs, is the final technical chapter. It begins by examining some elementary properties of two-level graphs. For example, it exhibits small examples (the square and the claw) that highlight the difference between two-level graphs and indifference graphs. It continues by showing again how the realization can be exploited, this time by improving on the minimum weight independent set algorithm for cocomparability graphs. Chapter 6 also examines how intimately two-levels graphs are related to other classes of graphs, such as bipartite permutation graphs and trapezoid graphs. The chapter concludes by studying the recognition problem for two-level graphs.

Finally, Chapter 7, the Conclusion, summarizes the thesis.

Two Notes About the Bibliography This thesis mentions many related results. For the most part, the bibliography cites the originators, unless a result is very well known. The rule of thumb is that a result is *very well known* if it is discussed in a standard textbook, such as *Introduction to Algorithms* by Cormen, Leiserson, and Rivest [CLR90]. In such a case, the bibliography normally cites the standard text, rather than the original.

Also, the thesis mentions some results in passing, for completeness, and I may not have been able to obtain the paper in question. Also, the bibliography may cite a result used by another researcher, but not directly by this thesis. In such cases, the bibliography cites the paper, but includes an annotation of the form "cited by [abc]", where "abc" is another paper that discusses and cites the result.

Chapter 2

Related Research

This chapter describes some of the context for the research described in this thesis, including other classes of graphs that are related to unit disk graphs. It begins (§2.1) by describing how and where unit disk graphs have arisen previously. This chapter also describes other kinds of intersection graphs (§2.2), perfect graphs (§2.3), and other kinds of proximity graphs (§2.4).

2.1 Unit Disk Graphs

Unit disk graphs are the subject of several algorithmic investigations. In particular, Clark, Colbourn, and Johnson [CCJ90] show that several well-known problems remain **NP**-complete for unit disk graphs (see Section 3.3 for details). Marathe, Breu, Hunt, Ravi, and Rosenkrantz [MHR92, MBH$^+$95] develop several approximation algorithms for unit disk graphs (again, see Section 3.3 for details).

Unit disk graphs are also simplified models for several applications. In general, they model the interaction of objects in a physical situation where the interaction (forces, visibility, or whatever) is "cut off" after a fixed distance. Sections 2.1.1 to 2.1.4 present some examples.

2.1.1 Cluster Analysis

Unit disk graphs arise as a conceptual mechanism in cluster analysis [Zah71, DH73, God88] when "dissimilarity" is measured by Euclidean distance. More precisely, the similarity of a set of objects $O = \{O_1, \ldots, O_n\}$ is typically described by an $n \times n$ dissimilarity matrix D that is not necessarily Euclidean nor even metric. Godehardt [God88] defines a "similarity" graph $\Gamma(\delta) = (O, E)$ where $(O_i, O_j) \in E$ if $D_{ij} \leq \delta$. Note that $G_\delta(O)$ is a unit disk graph if the O_i are points in the plane and $D_{ij} \stackrel{\text{def}}{=} \|O_i - O_j\|$. Godehardt mentions that the clusters produced by two well known techniques, single-linkage clusters and complete linkage clusters, correspond to the components and the cliques of the graph Γ, respectively. Efficient algorithms for constructing the components and a maximum clique of a unit disk graph are given in Sections 3.2.3 and 3.3.2 of this thesis.

2.1.2 Random Test Case

Johnson, Aragon, McGeogh, and Schevon [JAMS91] use a class of random graphs to test heuristics for the travelling salesman problem in arbitrary graphs. Their *random geometric graph* $U_{n,r}$ is, in the terminology of this thesis, the unit disk graph $G_r(P)$ where $|P| = n$, and where P is chosen randomly from a unit square[1]. Efficient algorithms for unit disk graphs could therefore be used to give exact solutions to various problems on random geometric graphs. These exact solutions could then be compared with those produced by the heuristic under evaluation. For example, Philips [Phi90] uses $U_{n,0.5}$ to evaluate his heuristic for finding the maximum clique, which he does by comparing it with another heuristic. The $O(n^{3.5} \log n)$ maximum clique algorithm presented in Section 3.3.2 (or the $O(n^{4.5})$ algorithm in [CCJ90]) would have allowed him to make a comparison with the exact solution in polynomial time.

[1]Johnson *et al.* generate P by picking $2n$ independent numbers uniformly from the interval $(0, 1)$ and viewing these as the coordinates of n points.

2.1.3 Molecular Graphics and Decoding Noisy Data

Bentley, Stanat, and Williams [BSW77] mention that merely listing the edges of a unit disk graph, a problem that they call "finding fixed-radius near neighbors", also has applications in molecular graphics and decoding noisy data. Their algorithm, along with others, is discussed in Section 3.2.3.

2.1.4 Radio Frequency Assignment

Suppose that a radio spectrum manager wishes to assign frequencies to a set of transmitters. Perhaps the simplest two-dimensional setting is that these transmitters all have the same power, and each has a fixed location on a uniform terrain so that their effective ranges are the same. Suppose further that the available frequencies, called *channels*, are a discrete set. Two transmitters must not be assigned the same channel if they might interfere, which they would do if they are within a fixed distance of one another. The manager's objective is to use the minimum number of channels.

In his review of spectrum management, Hale [Hal80] calls this problem the *F*D constrained cochannel assignment problem*. He formalizes it as follows. Given a finite set of points V in the plane, and a positive rational number δ, find an assignment $A : V \to \mathbf{Z}^+$ such that (1) $\max A(V)$ is as small as possible and (2) if $u, v \in V$, $u \neq v$, and $\|u - v\| \leq \delta$, then $A(u) \neq A(v)$. This is just the problem of finding an optimal colouring of a unit disk graph.

Hale's survey concludes with some relevant open questions:

> There is no graph coloring algorithm or heuristic which exploits the special structure of unit disk or disk graphs. There is no known intrinsic characterization of unit disk graphs (a reasonable forbidden subgraph characterization

seems out of the question, as a large list of infinite families of forbidden subgraphs continues to grow). What is the complexity of the clique problem ... for unit disk graphs?

These questions are at least partially resolved in this thesis, and elsewhere. In reverse order, the clique problem is in solvable in $O(V^{3.5} \log V)$ time (§3.3.2), UNIT DISK GRAPH RECOGNITION is **NP**-hard (§3.4), and there are simple polynomial time algorithms that colour unit disk graphs with no more than three times the chromatic number (§3.3.3).

The most comprehensive study of this frequency assignment (or unit disk graph colouring) problem is due to Gräf (see [Grä95] and §3.3.3). In his PhD thesis, Gräf also identifies and colours strip graphs (which he calls $\sqrt{3}/2$ stripe graphs).

2.2 Geometric Intersection Graphs

A graph G is an S-*intersection graph* if S is a family of sets, and there is a mapping $f : V(G) \rightarrow S$ such that $(u, v) \in E(G)$ if and only if $f(u) \cap f(v) \neq \emptyset$ for all vertices u and v in V. The function f is called an S-*realization* (or a representation, or a model) of G in terms of S.

Any graph is an intersection graph for *some* model. For example, model each vertex of a given graph by the set of edges incident with it. Geometric models promise to constrain the available intersection graphs that can be represented in this way. But even greatly constrained geometric models can generate many graphs. Any planar graph is an intersection graph of a set of curves in the plane. In fact, an intersection model made up solely of star-shaped polygons can be constructed from a straight line embedding[2] of a planar graph. The stars are concentric with the vertices, and their "arms" reach up

[2] Every planar graph has a straight line embedding (*cf.* [NC88]).

along each line segment corresponding to an edge.

More remarkably, every finite planar graph is the intersection graph of a set of interior-disjoint disks. Since such intersection graphs, also called *coin graphs*, are themselves clearly planar, this means that the class of finite planar graphs is equivalent to the class of coin graphs. This result was first demonstrated by Koebe in 1935. See Sachs's [Sac94] account for a more detailed history and for references.

Wegner [Weg67, CM78] proves that any graph is the intersection graph of a set of convex objects in $\mathbf{R}^3$. Unfortunately, Capobianco and Molluzzo [CM78], who cite Wegner's result, do not give the proof.

Researchers have studied numerous kinds of geometric intersection models other than disks and unit disks. In particular, graphs with the following one- and two-dimensional intersection models, some with efficient algorithms, can be found in the literature:

- unit intervals [Rob68a, LO93],
- intervals [FG65, GH64, BL76, GLL82, Kei85, Kei86, Ber86, BB86, RR88a, RR88b],
- squares [Ima82, BB86],
- rectangles [IA83, Kra94],
- line segments [IA86, KM94],
- strings [Kas80, Kra91a, Kra91b],
- circular arcs [Kas80, GLL82],
- chords of a circle [Kas80].

Recognition complexity questions have also been studied in the literature. In particular, there are polynomial time algorithms for recognizing intersection graphs on unit intervals [LO93], intervals [BL76], circular arcs [Tuc80], and chords of a circle [GHS89]. In his summary of graph algorithms [van90], van Leeuwen states wistfully that "It should probably be required of any class of graphs that is distinguished that its recognition problem is of polynomial-time bounded complexity." This desire notwithstanding, many interesting classes are difficult to recognize. In particular, it is **NP**-hard to recognize intersection graphs for strings [Kra91b], isothetic rectangles [Kra94], and segment graphs [KM94]. The complexity of recognizing unit square graphs was still open. However, Theorem 3.46 proves that this problem is **NP**-hard.

2.3 Perfect Graphs

Historically, there were two kinds of perfect graphs, both identified by Berge [Ber61]. A graph G is *χ-perfect* if the chromatic number χ of every induced subgraph $G(S)$ is the same as the size ω of its largest clique. More formally, a graph G is *χ-perfect* if $\omega(G(S)) = \chi(G(S))$ for all $S \subseteq V$. Clearly, $\chi(G(S)) \geq \omega(G(S))$ for all graphs, perfect or otherwise, since every vertex in a clique must get a different colour. A graph G is *α-perfect* if the number k of cliques required to cover any induced subgraph is the same as the size α of its maximum independent set. More formally, a graph G is *α-perfect* if $\alpha(G(S)) = k(G(S))$ for all $S \subseteq V$. Clearly, $k(G(S)) \geq \alpha(G(S))$ for all graphs since every clique can cover at most one vertex in an independent set. Clearly, too, a graph is χ-perfect if and only if its complement is α-perfect, since a clique in a graph is an independent set in its complement. A graph is *perfect* if it is both χ-perfect and α-perfect. In fact, a graph is χ-perfect if and only if it is α-perfect, as first conjectured by Berge [Ber61] and proved ten years later by Lovász [Lov72]:

Theorem 2.1 (The Perfect Graph Theorem [Ber61, Lov72]) *The following statements are equivalent for all graphs G:*

- $\omega(G(S)) = \chi(G(S))$ *for all* $S \subseteq V$,

- $\alpha(G(S)) = k(G(S))$ *for all* $S \subseteq V$,
- $\omega(G(S))\alpha(G(S)) \geq |S|$ *for all* $S \subseteq V$.

Grötschel, Lovász, and Schrijver [GLS84] show that these invariants ($\alpha(G)$, $\omega(G)$, $\chi(G)$, and $k(G)$) can be computed in polynomial time for perfect graphs. Their algorithms use the ellipsoid method, a general technique that has also been used to solve linear programming in polynomial time (see [GLS84] for details and references). Grötschel *et al.* do not recommend their algorithms for practical use, due to numerical stability problems with the ellipsoid method. Furthermore, the complexity of PERFECT GRAPH RECOGNITION is unknown, though Grötschel *et al.* show that it is in **co-NP**. Consequently, there are many results on subclasses of perfect graphs in the literature (see in particular Golumbic's book [Gol80]). There are many such subclasses. For example, bipartite graphs are clearly χ-perfect; assuming a bipartite graph has at least one edge, both its clique number and chromatic number is 2, and it is trivial to find such a maximum clique or such a colouring. Finding a maximum independent set of a bipartite graph is not that easy, but can be accomplished in polynomial time by matching (*cf.* §3.3.2).

In this thesis, cocomparability graphs (see Section 2.3.1 and Chapter 4), which are perfect, are especially relevant. The family of cocomparability graphs properly includes the family of trapezoid graphs (see Section 6.4.1). In turn, the family of trapezoid graphs properly includes the families of permutation and interval graphs (see below for definitions). Finally, the family of interval graphs properly include the family of indifference graphs.

On the other hand, unit disk graphs are not perfect. For example, C_5 is a unit disk graph (as is any cycle) but it is not perfect since $\omega(C_n) = 2$ and $\chi(C_n) = 3$ for all odd $n \geq 5$. However, strip graphs are perfect since they are cocomparability graphs.

2.3.1 Cocomparability Graphs

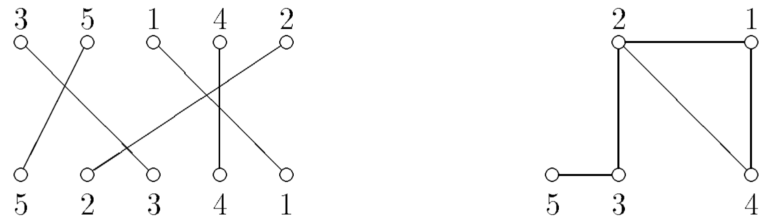
An undirected graph $G = (V, E)$ is a *comparability graph* if there exists a transitive orientation of its edges. We can also say that a comparability graph is a *transitively orientable* graph. Recall from Section 1.3.2 that a *transitive orientation* of an undirected graph $G = (V, E)$ is an oriented subgraph $\vec{G} = (V, A)$ where $E = A + A^{-1}$ and A is transitive. Note that transitive orientations are acyclic since A is asymmetric. Conversely, we can think of comparability graphs as being generated by partially ordered sets. The edges of the *comparability graph* $G = (V, E)$ *corresponding to a strict poset* $(V, <)$ are given by $E = \{(u, v) : u < v \text{ or } v < u\}$. A graph is a *cocomparability graph* if its complement is a comparability graph. Both comparability graphs and cocomparability graphs have been extensively studied (*cf.* [Gol80, Möh85]). Both classes include permutation graphs as proper subclasses, and cocomparability graphs properly include interval graphs and indifference graphs (*cf.* [Duc84]).

Comparability graphs (and therefore cocomparability graphs also) can be oriented in $O(V^2)$ and recognized in $O(M(V))$ time [Spi85, Spi94]. Previous algorithmic work on cocomparability graphs relevant to this thesis includes polynomial time algorithms for several domination problems ([KS93], but see §4.2 for improved time complexities), and a cubic time algorithm for the Hamiltonian cycle problem [DS94]. There are also algorithms for comparability graphs [Möh85], so that the complementary problem can be solved on cocomparability graphs. Once a vertex-weighted comparability graph has been transitively oriented, one can extract a maximum weight clique in $O(V^2)$ time[3], and a maximum weight independent set in $O(V^3)$ time [Möh85].

[3]This algorithm for maximum weight clique cannot be used to find a minimum weight *maximal* clique. See §4.2.4 for an explanation and an algorithm that finds a minimum (and maximum) weight maximal clique in $O(M(V))$ time.

Permutation Graphs

A graph $G = (V, E)$ is a *permutation graph* if there is some pair of permutations (ordered lists) P and Q of the vertices such that u and v are adjacent if and only if their order differs in the two permutations. The permutations P and Q are said to *realize* the permutation graph G, and make up a *permutation realization* or *permutation model*. For example, if $P = (3, 5, 1, 4, 2)$ and $Q = (5, 2, 3, 4, 1)$ model some permutation graph $G = (V, E)$, then $(3, 5) \in E$ but $(3, 1) \notin E$. We can also think in terms of a *permutation diagram*; lay out the permutations on two parallel rows and connect corresponding vertices with line segments. Then u and v are adjacent if and only if their corresponding line segments cross; see Figure 2.1, for example.



(a) (b)

Figure 2.1: A permutation diagram (a) and its corresponding permutation graph (b). Here, $P = (3, 5, 1, 4, 2)$ and $Q = (5, 2, 3, 4, 1)$.

To simplify the notation, the first permutation P is traditionally (*cf.* [Gol80]) written as a relabelling of the vertices $P = (1, 2, \ldots, |V|)$. The second permutation Q is written as a list $Q = (\pi(1), \pi(2), \ldots, \pi(|V|)$, where $\pi : V \to V$ is a permutation (that is, a bijection) on V. Then $(u, v) \in E$ if and only if $(u - v)(\pi^{-1}(u) - \pi^{-1}(v)) < 0$ (think of $\pi^{-1}(v)$ as the "location" of v in Q). We can therefore think of π as the *defining permutation* (corresponding to permutation P) for the permutation graph G. Spinrad has shown that a defining permutation for a given permutation graph can be found in $O(V^2)$ time [Spi85].

The complement of a permutation graph is also a permutation graph. It is easy to verify that if π is a defining permutation for G, then the *reverse* π^r of π, where $\pi^r(i) = \pi(|V|+1-i)$ for all i, is a defining permutation for $\overline{G}$. Since permutation graphs are comparability graphs, it follows that permutation graphs are also cocomparability graphs. In fact, this latter observation characterizes permutation graphs. That is, a graph G is a permutation graph if and only if G and $\overline{G}$ are comparability graphs [PLE71]. Recall that the *dimension* of a partial order P is the minimum number of linear orders whose intersection is P. If G is the comparability graph of a partial order P, then P has dimension at most 2 if and only if $\overline{G}$ is transitively orientable [DM41] (that is, $\overline{G}$ is also a comparability graph). It follows that G is a permutation graph if and only if the partial order underlying any transitive orientation of G has dimension at most 2.

There has been considerable interest in solving dominating set problems on permutation graphs [FK85, BK87, CS90a, TH90, AR92]. These problems are discussed in Section 4.2, which also summarizes their complexity on permutation graphs. That section presents new polynomial time algorithms for these dominating problems on cocomparability graphs, which properly includes permutation graphs, as mentioned above. Permutation graphs also play a role in Chapter 6, in which bipartite permutation graphs appear in strip graphs.

Function Graphs

Cocomparability graphs may also be characterized as intersection graphs. Let F be a family of continuous functions $f_i : [0,1] \rightarrow \mathbf{R}$. Say that two functions f_i and f_j *intersect* if there is a value $x \in [0,1]$ such that $f_i(x) = f_j(x)$, that is, the images of $[0,1]$ under f_i and f_j intersect. *Function graphs* are the intersection graphs of such families F. Golumbic, Rotem, and Urrutia [GRU83] show that a graph is a function graph if and only if it is a cocomparability graph.

Since all partial orders corresponding to a comparability graph have the same dimension [TJS76], this number can equally well serve as the dimension of the comparability graph. Golumbic *et al.* show that a cocomparability graph is the concatenation of $k-1$ permutation diagrams, where k is the dimension of the graph's complement. For example, permutation graphs (and their complements) have dimension 2. This also characterizes permutation graphs; they are exactly the intersection graphs of linear functions.

Interval graphs are also the concatenation of permutation diagrams, since they are cocomparability graphs. However, interval graphs cannot be characterized as the concatenation of some finite number of permutation graphs. For example, C_4 is a permutation graph, but it is not an interval graph.

However, we can characterize interval graphs with a class of piecewise-linear functions, made up of three linear pieces. Let $G = (V, E)$ be an interval graph where $n = |V|$, and let $I = \{[l_1, r_1], [l_2, r_2], \ldots, [l_n, r_n]\}$ be an interval realization of G. We can assume without loss of generality that $l_v, r_v \in \{1, 2, \ldots, 2n\}$. The function $f_v : [0, 1] \to \mathbf{R}$ corresponding to the interval $[l_v, r_v]$, where $l_v \leq r_v$, is given by:

$$f_v(x) = \begin{cases} l_v & \text{if } 0 \leq x \leq \frac{l_v}{2n+1} \\ (2n+1)x & \text{if } \frac{l_v}{2n+1} < x < \frac{r_v}{2n+1} \\ r_v & \text{if } \frac{r_v}{2n+1} \leq x \end{cases}$$

This process is illustrated in Figure 2.2, which shows the "construction line"

$$y = (2n+1)x$$

for clarity. If two functions intersect, then they do so on the construction line, and mimic the original intervals. Clearly then, two intervals intersect if and only if the corresponding functions intersect.

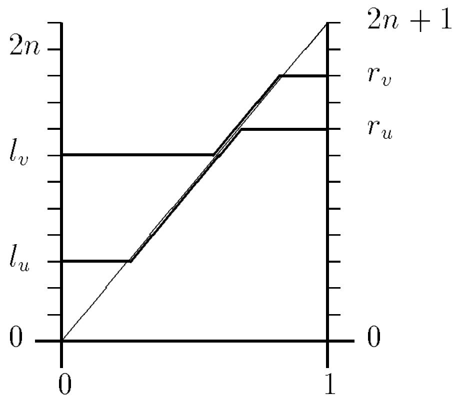


Figure 2.2: Functions for intervals $[l_u, r_u]$ and $[l_v, r_v]$.

2.3.2 Indifference Graphs

Roberts [Rob68a] named indifference graphs after the notion of "indifference" from psychology and economics. The subsection introduces the idea of indifference graphs, and views them as one-dimensional unit disk graphs.

Can an individual's preference within a finite set of options be assigned a scalar measure? Such a measure f would map the options into real numbers. To a first approximation, the individual would prefer option a to b if and only if $f(a) > f(b)$. This approximation would imply that indifference corresponds to equality and is therefore transitive. Roberts [Rob78] cites several sources that argue against the transitivity of indifference. To model indifference more accurately, he suggests that the preference measure have the property that, if the individual is indifferent between a and b, then $|f(a) - f(b)| < \delta$.

Definition 2.2([Rob68a]) A graph $G = (V, E)$ is an *indifference graph* if there is a mapping $f : V \to \mathbf{R}$ of its vertices to the reals such that $(u, v) \in E$ if and only if $|f(u) - f(v)| \leq 1$. ∎

Although Definition 2.2 is sufficient, indifference graphs can also be defined by any of the characterizations in Theorem 2.3. All uncited characterizations below are due to Roberts [Rob68a]. Definitions of the emphasized terms in the characterizations follow the theorem.

Theorem 2.3 *Let $G = (V, E)$ be a graph. Then the following are equivalent:*

1. *G is an indifference graph.*
2. *There is a* semiorder $(V, <)$ *such that $(u, v) \in E$ if and only if neither $u < v$ nor $v < u$, for all u, v in V.*
3. *G is a* unit interval graph
4. *G is a $K_{1,3}$-free* interval graph.
5. *G is a* proper interval *graph.*
6. *([Rob78] page 33) The adjacency matrix of G has the* consecutive-ones property.
7. *G is chordal and none of the graphs in Figure 2.3 are induced subgraphs.*
8. *[Duc84] The closed neighbourhoods of G constitute the edges of an* interval hypergraph.
9. *[Duc84] G admits an orientation $\vec{G}$ that satisfies both:*
   (a) *$\vec{G}$ has no cycle,*
   (b) *$G(\mathrm{N}^+(v))$ and $G(\mathrm{N}^-(v))$ are complete graphs.*
10. *[Jac92] G is an* astral triple-*free graph.*
11. *[Duc79] There exists a linear ordering of V such that, if $u < v < w$ and $(u, w) \in E$, then both (u, v) and (v, w) are in E.*

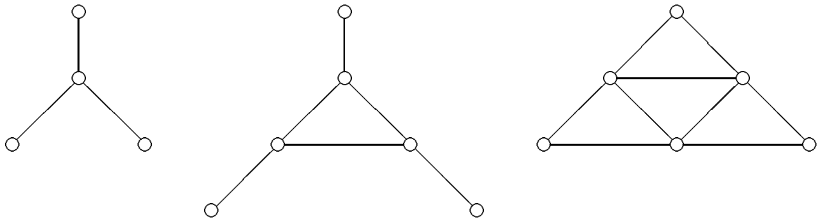


Figure 2.3: Forbidden indifference graphs. These graphs, together with C_n for $n \geq 4$, make up the forbidden subgraph characterization of indifference graphs.

astral triple Three vertices in a graph such that, between any two of them, there is path P that avoids the third vertex v, and no two consecutive vertices in P are both adjacent to v.

consecutive-ones property A binary matrix is said to have the *consecutive-ones property* if it is possible to permute the rows so that the ones in each column are consecutive.

interval graph The intersection graph of a set of intervals on a linearly ordered set.

interval hypergraph A (not necessarily binary) relation $H = (V, E)$ for which there exists a linear order on the vertices V such that every element (*edge*) in E is an interval in the order.

proper interval graph The intersection graph of a set of intervals on a linearly ordered set, none of which is contained by (is a subset of) any other.

semiorder An irreflexive relation $(A, <)$ is a *semiorder* if for all $x, y, z, w \in A$: $x < y$ and $z < w$ implies $x < w$ or $z < y$; and $x < y$ and $y < z$ implies $x < w$ or $w < z$.

unit interval graph The intersection graph of a set of equal-sized (unit length) intervals on the real line.

Looges and Olariu [LO93] have developed several optimal algorithms for indifference graphs. In particular, they show how to recognize an indifference graph $G = (V, E)$ in $O(V + E)$ time. Their recognition algorithm returns an efficient representation of the ordering from Theorem 2.3.11. Using this representation, Looges and Olariu colour the graph, find a shortest path between two vertices, compute a Hamiltonian path, and compute a maximum matching, each in $O(V)$ additional time.

It is clear from the definition of indifference graphs (Definition 2.2) that the class *indifference graphs* is the special case of τ-strip graphs where $\tau = 0$ (*cf.* Definition 1.3). It would be beneficial for algorithm design if some or all of the properties for indifference graphs in Theorem 2.3 were preserved for τ-strip graphs as τ increases. However, such preservation does not seem to be the case. For example both the square and the claw are τ-strip graphs for all $\tau > 0$. In particular, the algorithms by Looges and Olariu do not seem to generalize, since the closest analogue to Theorem 2.3.11 that applies to τ-strip graphs is the spanning order (which characterizes the more general class of cocomparability graphs) discussed in Chapter 4.

Roberts [Rob68b] generalizes the notion of indifference to higher dimensions by defining the *cubicity* of a graph to be the smallest dimension k for which there exists a function $f:\ V \rightarrow \mathbf{R}^k$, such that $(u, v) \in E$ if and only if $\|f(u) - f(v)\|_\infty \leq 1$. The corresponding notion for the L_2 metric is called the *sphericity* [Hav82a, Fis83] of a graph. Clearly, the graphs that have cubicity or sphericity at most one are precisely the indifference graphs. The notion of sphericity is well defined, as Maehara's theorem shows.

Theorem 2.4 ([Mae84]) *Every graph of order n is isomorphic to some space graph* [unit sphere graph] *in n-space.*

The sphericity (respectively, cubicity) of a graph is just the smallest dimension in which it is a unit disk graph—generalized with respect to dimension—under the L_2 (respectively,

L_∞) metric. The (not generalized) unit disk graphs correspond exactly to graphs with sphericity at most 2.

How is sphericity related to cubicity? Havel [Hav82a, Fis83] shows that cubicity can exceed sphericity. For example the cubicity and sphericity of $K_{1,5}$ are 3 and 2 respectively. In fact, Havel shows that there are finite graphs of sphericity 2 that have arbitrarily large cubicity. Fishburn [Fis83] shows that sphericity can exceed cubicity for cubicity 2 and 3 but leaves the question open for larger cubicity. Maehara [Mae86] answers this question by showing that there is a complete bipartite graph with sphericity exceeding cubicity for every value of cubicity at least 6.

2.3.3 Grid Graphs

A *grid graph* is a node-induced finite subgraph of the infinite grid [IPS82]. Equivalently therefore, a grid graph is a unit disk graph that has a realization with integer coordinates. More carefully, a graph G is a *grid graph* if there is a function $f : G(V) \rightarrow \mathbf{Z}^2$ such that, $(u, v) \in E$ if and only if $\|f(u) - f(v)\| \leq 1$ for all $u, v \in V$. Some authors [Had77, BC87] prefer to define grid graphs as (not necessarily induced) subgraphs of the infinite grid (let us call these *grid subgraphs*). For example, the tree in Figure 2.4 is a grid subgraph, but not a grid graph, since every grid realization of this graph must put two leaves within unit distance of one another.

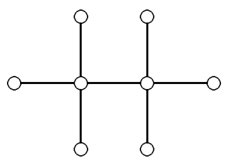


Figure 2.4: A grid subgraph that is not a grid graph.

Grid graphs arise in several applications, including integrated circuit design (*cf.* [Had77]).

The infinite grid represents the possible conducting material. A chip isolates a portion of the grid, effectively inducing a rectangular subgraph on the grid. The graph is further complicated by electronic components on the chip, which remove any covered vertices and edges from the graph. A typical problem in this context is to lay out (embed) an interconnection pattern (a graph) on the chip (in the corresponding grid graph), possibly by dilating (subdividing) edges in the pattern, so as to minimize the size of its layout.

Since grid graphs (and grid subgraphs) are bipartite, and bipartite graphs are perfect, grid graphs are also perfect and subject to the same efficient algorithms (see Section 2.3). On the other hand, several familiar problems that are **NP**-complete for arbitrary graphs remain **NP**-complete for grid graphs, including finding a Hamiltonian path [IPS82] and finding an optimal Steiner tree [GJ77].

GRID SUBGRAPH RECOGNITION is **NP**-complete [BC87], even for grid subgraphs that are binary trees [Gre89]. The following simple reduction shows that GRID GRAPH RECOGNITION is also **NP**-complete. Let $G = (V, E)$ be an instance of GRID SUBGRAPH RECOGNITION. Create a graph $G' = (V', E')$ from G by replacing each edge in E with the *edge simulator* graph shown in Figure 2.5. Since each edge simulator has

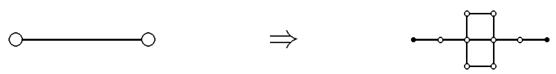


Figure 2.5: Simulating edges in grid subgraphs with grid graphs.

a unique induced embedding in the grid (up to rotation, reflection, and translation), it follows that G' is a grid graph if and only if G is a grid subgraph. Figure 2.6 shows an embedded grid subgraph and the embedded grid graph that results from this reduction.

In his PhD thesis [Grä95], Gräf generalizes the notion of grid graphs by allowing disks of larger than unit diameter. More formally, he defines the class of UD_d graphs

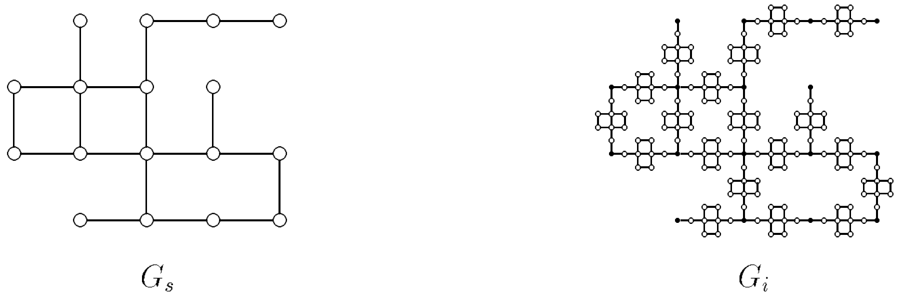


Figure 2.6: The grid graph G_i corresponding to the grid subgraph G_s.

to be the intersection graphs of disks with diameter $d \in \mathbf{Z}^+$ and centres in $\mathbf{Z}^2$. Clearly, all UD_d graphs are unit disk graphs, and UD_1 is equivalent to the class of grid graphs. Gräf shows that each UD_d graph $G = (V, E)$ has a realization that can be encoded with $O(V \log(dV))$ bits, so that UD_d GRAPH RECOGNITION is in **NP**. By comparison, UNIT DISK GRAPH RECOGNITION is not known to be in **NP**. He shows that UD_2 graph recognition is **NP**-complete by reducing it from grid graph recognition. He also conjectures that UD_d recognition is **NP**-complete for all fixed d. Again by comparison, Section 3.4 shows that UNIT DISK GRAPH RECOGNITION is **NP**-hard. Finally, Gräf shows that UD_2 GRAPH 3-COLOURABILITY is **NP**-complete. Since $\text{UD}_d \subseteq \text{UD}_{kd}$ for all positive integers d and k, it follows that GRAPH 3-COLOURABILITY is **NP**-complete for all UD_d where d is even.

2.4 Proximity Graphs

Two vertices are adjacent in a unit disk graph if their realizations satisfy a certain proximity condition: they must be within unit distance of one another. Other proximity conditions lead to other kinds of graphs.

2.4.1 The Delaunay Triangulation Hierarchy

The *Euclidean minimum spanning tree* [Zah71, SH75, Yao82, AESW90] *MST(P)* of a set of points P in the plane is a minimum spanning tree of the complete graph $K(P)$ with edge weights equal to the Euclidean distance between endpoints.

The *relative neighbourhood graph* [Tou80, Sup83] $RNG(P) = (P, E_{RNG})$ of P has an edge (p, q) between two vertices if and only if the lune through p and q does not contain any other point in P. That is, $(p, q) \in E_{RNG}$ if and only if $\|p - q\| \leq \|p - r\|$ and $\|p - q\| \leq \|q - r\|$ for all r not equal to p or q.

Two vertices p and q of the *Gabriel graph* [GS69, MS80] $GG(P) = (P, E_{GG})$ are adjacent if and only if the smallest disk through p and q does not contain any other point in P. That is, $(p, q) \in E_{GG}$ if and only if $\|p - q\| \leq \|p - r\| + \|q - r\|$ for all r not equal to p or q.

Perhaps the best known proximity graph is the Delaunay graph. Three vertices p, q, and r are pairwise adjacent in the *Delaunay graph* [Del34] $DT(P)$ if and only if the smallest disk through all three does not contain any other point in P. Since the (straight line) plane graph corresponding to $DT(P)$ partitions the plane into triangles and an external face, the Delaunay graph is usually called the Delaunay triangulation. An alternative definition involves the Voronoi diagram [Vor08, PS85]. The *Voronoi polygon* or *Voronoi cell* associated with a point p is the set of points in the plane that are closer to p than any other point in P. The *Voronoi diagram* is the partition of the plane induced by the Voronoi polygons associated with the points in P. Define an edge in E_{DT} to be a pair of points (p, q) if and only if the Voronoi polygons associated with p and q share an edge. That is, the Delaunay triangulation is the straight line geometric dual of the Voronoi diagram.

These four graphs are related as follows [PS85]:

$$MST(P) \subseteq RNG(P) \subseteq GG(P) \subseteq DT(P).$$

The number of edges in each of these graphs is linear in the number of vertices. Historically at least, there has been little emphasis on recognizing these graphs. However, Eades and Whitesides [EW94] recently showed that recognizing Euclidean minimum spanning trees is **NP**-hard. There has been more emphasis on constructing these graphs from a given point set P; each graph can be constructed in $O(P \log P)$ time ([PS85, Sup83, MS80, PS85] respectively).

2.4.2 Sphere of Influence Graphs

Let P be a set of points in the plane and let $D = \{D_p : p \in P\}$ be a corresponding set of disks, where D_p is the largest disk centered on p whose interior is empty of any point in P other than p. That is, $radius(D_p) = \min\{\|p - q\| : q \in P \text{ and } q \neq p\}$ for every $p \in P$. Avis and Horton [AH85] define the *(closed) sphere of influence graph* $\hat{G}(P)$ to be the intersection graph $\Omega(D)$ of the disks D. Clearly, the sphere of influence graph is a disk graph. They show that the sphere of influence graph is neither a subgraph nor a supergraph of any of the minimum spanning tree, the relative neighbourhood graph, the Gabriel graph, or the Delaunay triangulation. They also show that $\hat{G}(P)$ has at most $29|P|$ edges and that the graph can be constructed in $\Theta(P \log P)$ time, which they show to be optimal.

There is a subgraph relationship between sphere of influence graphs and unit disk graphs. In particular,

$$G_{\delta_1}(P) \subseteq \hat{G}(P) \subseteq G_{\delta_2}(P),$$

where $\delta_1 = 2 \cdot \min\{radius(D_p) : p \in P\}$ and $\delta_2 = 2 \cdot \max\{radius(D_p) : p \in P\}$. This also can be stated in terms of intersection graphs. The intersection graph $\Omega(C_{\min})$, where

$C_{\min}$ is a set of equal-diameter disks, is a subgraph of $\Omega(D)$, where D is a set of disks with varying diameters, concentric with $C_{\min}$, each of which is no smaller than the disks in $C_{\min}$. Similarly, the intersection graph $\Omega(D)$ of a set of disks D with varying diameters, is a subgraph of $\Omega(C_{\max})$, where $C_{\max}$ is a set of equal-diameter disks, concentric with D, each of which is no smaller than the largest disk in D.

Unit disk graphs share the property with sphere of influence graphs that they do not fall nicely into the Delaunay triangulation hierarchy. Nevertheless, one of the algorithms in Section 3.2.3 efficiently constructs (enumerates the edges of) a unit disk graph by thresholding the Delaunay triangulation.

Chapter 3

Unit Disk Graphs

How does the geometry of intersecting unit disks affect the associated intersection graphs? To address this question, Section 3.1 makes some basic observations about unit disk graphs that are exploited throughout the thesis. Section 3.2.1 then shows how to exploit a geometric realization of a unit disk graph to find least vertex-weight paths. Section 3.2.2 solves some more fundamental tasks: given a vertex $v \in V$, report a vertex adjacent to v (LIST1(v)), and delete v (MARK(v)). More specifically, it shows how to execute a sequence of $O(V)$ calls to LIST1 and MARK in $O(V \log V)$ time and $O(V)$ space. Section 3.2.3 shows how to construct a unit disk graph from its geometric realization (that is, from $|V|$ points in the plane) in $O(V \log V + E)$ time. Recall that efficient construction has historically been of interest for other kinds of proximity graphs (§2.4).

Section 3.3.2 shows how to use a realization to find a maximum clique in polynomial time. Unfortunately, the geometry is not always so exploitable. The rest of Section 3.3 shows that several familiar problems remain **NP**-hard for unit disk graphs. In proving some of these (INDEPENDENT SET and CHROMATIC NUMBER), this section draws a connection between unit disk graphs and planar graphs with no degree exceeding 4.

Finally, we will see in Section 3.4 that even unit disk graph recognition is **NP**-hard. Furthermore, penny graph, bounded-diameter-ratio coin graph, and bounded-diameter-ratio disk graph recognition are all **NP**-hard, even for square disks and coins. That is, both 2-SPHERICITY and 2-CUBICITY are **NP**-hard.

3.1 Basic Observations

This section makes some elementary observations about disk graphs and unit disk graphs that will prove useful later in the thesis. In particular, this section proves that the star $K_{1,6}$ is forbidden, and that triangle-free disk graphs are planar. We will also see that all strip graphs are cocomparability graphs.

Lemma 3.1 (The Star Lemma) *There are no induced stars with degree greater than 5 in any unit disk graph, and no induced stars with degree greater than 4 under the L_1 and L_∞ metrics.*

Proof: At most five Euclidean disks can pack around a central disk without intersecting one another. More precisely, let $f : V \to \mathbf{R}^2$ be a realization of a unit disk graph $G = (V, E)$. Let u be any vertex in V, and let $I \subseteq \mathrm{Adj}(u)$ be a set with more than five vertices, i.e., $|I| \geq 6$. Then there must be two vertices v and w in I such that the angle between the segments $s_v = (f(u), f(v))$ and $s_w = (f(u), f(w))$ is at most 60 degrees. But then, since the segments s_v and s_w have at most unit length, it follows that the segment $(f(v), f(w))$ also has at most unit length. Therefore $(v, w) \in E$ and $G(I \cup \{v\})$ is not isomorphic to $K_{1,|I|}$.

Similarly, disks under the L_1 and L_∞ metrics are closed square boxes. At most four such boxes can pack around a central box without intersecting one another. ▮

Corollary 3.1.1 *The star $K_{1,6}$ is not an induced subgraph of any unit disk graph.*

Lemma 3.2 *In every unit disk graph, there is a vertex v that has degree at most 3 in any induced star. Similarly, in every unit disk graph under the L_1 and L_∞ metrics, there is a vertex v that has degree at most 2 in any induced star.*

Proof: Let $f : V \to \mathbf{R}^2$ be a realization of a unit disk graph $G = (V, E)$. Let $v \in V$ be a vertex such that $f(v)$ is an extreme point, a leftmost point, for example. Then the neighbourhood of v lies in a half plane through v. The rest of the proof is similar to that of the Star Lemma. ∎

3.1.1 Connections with Planarity

Recall from Section 2.2 that Koebe showed that every finite planar graph is the intersection graph of a set of interior-disjoint disks. So every finite planar graph is a disk graph. However, there are disk graphs (without the interior-disjoint restriction) that are not planar. For example, disk graphs may contain arbitrarily large cliques. This subsection shows (Theorem 3.4) that if a disk graph contains only very small cliques (with at most two vertices), then it is planar. The converse is clearly not true, as demonstrated by a triangular packing of unit disks, which generates a planar disk graph with many triangles (cliques with three vertices).

Lemma 3.3 *Let $f : V \to \mathbf{R}^2$ (locations) and $r : V \to \mathbf{R}$ (disk radii) be a realization of a disk intersection graph $G = (V, E)$. Let (a, b) and (c, d) be edges in E with distinct endpoints. If the line segments $(f(a), f(b))$ and $(f(c), f(d))$ cross, then the subgraph induced by $\{a, b, c, d\}$ contains a triangle.*

Proof: For readability, let v denote also $f(v)$ for $v \in \{a, b, c, d\}$. Suppose that (a, b) and (c, d) cross at some point e, as shown in Figure 3.1. Then the sum of each pair of opposite

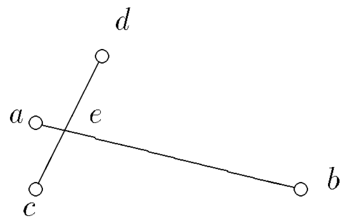


Figure 3.1: If two segments cross in a disk graph realization, then their endpoints induce a triangle.

sides of the quadrilateral $acbd$ is at most the sum of its diagonals. More precisely, let uv denote the Euclidean distance $\|u - v\|$ between points u and v. Then

$$\begin{aligned} ac + bd &\leq (ae + ec) + (be + ed) \\ &= (ae + be) + (ec + ed) \\ &= ab + cd. \end{aligned}$$

Since $(u, v) \in E$ if and only if $uv \leq r(u) + r(v)$, it follows that

$$\begin{aligned} ac + bd &\leq ab + cd \\ &\leq r(a) + r(b) + r(c) + r(d) \\ &= (r(a) + r(c)) + (r(b) + r(d)). \end{aligned}$$

Therefore, either $ac \leq r(a) + r(c)$ or $bd \leq r(b) + r(d)$. That is, either $(a, c) \in E$ or $(b, d) \in E$. Similarly, $ad + bc \leq ab + cd$, so that either $(a, d) \in E$ or $(b, c) \in E$. Any of these four possibilities implies a triangle. ∎

Theorem 3.4 *Every triangle-free disk graph is planar.*

Proof: Let $f : V \rightarrow \mathbf{R}^2$ (locations) and $r : V \rightarrow \mathbf{R}$ (disk radii) be a realization of a disk intersection graph $G = (V, E)$. If, for some pair of edges (a, b) and (c, d) in E with distinct endpoints, the line segments $(f(a), f(b))$ and $(f(c), f(d))$ cross, then the endpoints induce a triangle in G by Lemma 3.3. Otherwise, no such line segments cross, and the graph is planar by definition. ∎

Corollary 3.4.1 *No disk graph has an induced subgraph homeomorphic to $K_{3,3}$.*

Proof: Suppose some disk graph has an induced subgraph homeomorphic to $K_{3,3}$. Then there is a disk graph homeomorphic to $K_{3,3}$, since any class of intersection graphs is closed under taking induced subgraphs. This graph is clearly not planar. However, all graphs homeomorphic to $K_{3,3}$ are triangle-free. Therefore G is planar by Theorem 3.4, a contradiction. ▮

On the other hand, there are K_4-free disk graphs that are not planar. For example, the graph in Figure 3.2 is homeomorphic to K_5. Note that although Figure 3.2 is K_4-

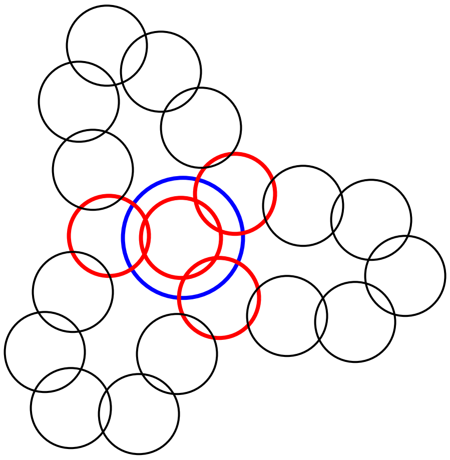


Figure 3.2: A K_4-free disk graph homeomorphic to K_5

free, it is naturally (by Theorem 3.4) not triangle-free . Figure 3.2 is actually a *unit* disk graph. To see this, notice that in the realization shown, there is exactly one disk that is larger than unit size. Simply shrink it about its centre until it is of unit size. This will make it less visible in the drawing, but will not affect its adjacency.

3.1.2 Every τ-Strip Graph is a Cocomparability Graph

Definition 3.5 A directed orientation $\vec{G} = (V, \vec{E})$ of the complement of a τ-strip graph G, and a realization $f : V(G) \to \mathbf{R} \times [0, \tau]$ are *compatible* if they satisfy $(u, v) \in \vec{E}$ if and only if $(u, v) \notin E$ and $x_f(u) < x_f(v)$ for all $u, v \in V$. ▮

Note that $\vec{G}$ really is an orientation since, for every $(u, v) \in \overline{E}$, either $x_f(u) < x_f(v)$ or $x_f(v) < x_f(u)$, but not both. That is, $\vec{G}$ is asymmetric, as required. Notice that there is precisely one compatible orientation for every realization, but that there might be several realizations that are compatible with any orientation.

Lemma 3.6 *For every strip thickness $\tau \in [0, \sqrt{3}/2]$, the compatible orientation associated with any τ-strip realization of a graph is a transitive orientation (i.e., a strict partial order).*

Proof: Let $G = (V, E)$ be a strip graph, f be a strip realization, and $\vec{G} = (V, \vec{E})$ the compatible orientation. Then $\vec{G}$ is transitive. For if $(a, b) \in \vec{E}$ and $(b, c) \in \vec{E}$ then $x_f(a) < x_f(b) < x_f(c)$. Furthermore, since $\|f(b) - f(a)\| > 1$ and $0 \leq \tau \leq \sqrt{3}/2$, we have

$$\begin{aligned} x_f(b) - x_f(a) &> \sqrt{1 - (y_f(b) - y_f(a))^2} \\ &\geq \sqrt{1 - \tau^2} \\ &\geq 1/2. \end{aligned}$$

Similarly, $x_f(c) - x_f(b) > 1/2$. Therefore,

$$\begin{aligned} x_f(c) - x_f(a) &= (x_f(c) - x_f(b)) + (x_f(b) - x_f(a)) \\ &> 1/2 + 1/2 \\ &= 1. \end{aligned}$$

It follows that $(a, c) \in \overline{E}$ and $(a, c) \in \vec{E}$.

By a similar proof, compatible orientations for L_1 strip graphs (for which $0 \leq \tau \leq 1/2$) are also transitive. ∎

Theorem 3.7 *Strip graphs form a subclass of cocomparability graphs.*

Proof: Follows immediately from Lemma 3.6. ∎

Theorem 3.8 *For every strip thickness $\tau > \sqrt{3}/2$, there is a τ-strip graph that is not a cocomparability graph.*

Proof: We will construct C_5 in a "thick" strip. Since cocomparability graphs are perfect but C_5 is not, it follows that C_5 is not a cocomparability graph. Chapter 4 proves the stronger claim that *no* induced cycle (odd or even) on five or more vertices is a cocomparability graph (Theorem 4.6).

Let y be any value that satisfies $\sqrt{3}/2 < y \leq \min\{\tau, \sqrt{15}/4\}$. Construct five points $P = \{p_1, p_2, p_3, p_4, p_5\}$ in the strip, where

$$\begin{aligned} p_1 &= (-3/4, 0), \\ p_2 &= (-1/2, y), \\ p_3 &= (1/2, y), \\ p_4 &= (3/4, 0), \\ p_5 &= (0, 0), \end{aligned}$$

as shown in Figure 3.3. Then the unit disk graph $G(P)$ generated by P is isomorphic to

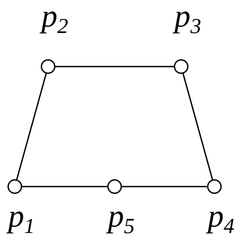


Figure 3.3: A set of five points that generate an induced cycle.

C_5 (although this is clearly not the only way to realize C_5 in a suitably thick strip). ∎

3.2 Exploiting a Geometric Model

3.2.1 Least Vertex-Weight Paths in Unit Disk Graphs

Finding short paths is a natural operation on graphs, see ([CLR90] pages 514–578) for example. This section develops an algorithm that finds a path (that minimizes the sum of the weights of its vertices) from a set of source vertices to all other vertices in a unit disk graph. That is, it solves the *multiple-source least vertex-weight path problem* on unit disk graphs.

Any algorithm for finding a least edge-weight path must examine at least $\Omega(E)$ edges, so that $\Omega(E)$ is a lower bound for the algorithm's run time. To see this, consider an algorithm on the bipartite graph in Figure 3.4 in which all edges but one, to be determined

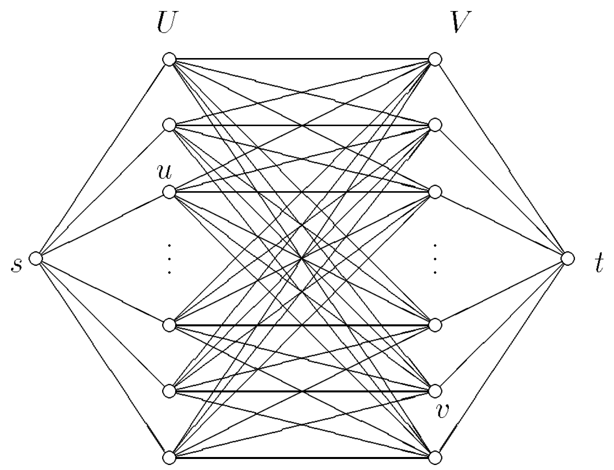


Figure 3.4: Adversary argument for shortest path

by an adversary, have weight 1. Suppose the algorithm fails to examine some edge (u, v) from the $|U| \cdot |V| = \Theta(E)$ edges between sets U to V. The adversary then makes (u, v) the 0-weight edge. Therefore, the algorithm clearly fails to find the shortest path (s, u, v, t) from s to t.

This adversary argument does not hold for least vertex-weight paths. If we could implicitly represent the edges, then we might be able to design algorithms whose complexities do not depend on the number of edges. This section does so by exploiting a geometric representation for unit disk graphs. In particular, it solves the *multiple-source least vertex-weight path problem* for unit disk graphs with nonnegative vertex-weight in $O(V \log V)$ time, where the unit disk graph $G = (V, E)$ is represented by a realization $f : V \to \mathbf{R}^2$. We will derive the final algorithm by modifying Dijkstra's algorithm in several phases.

By way of background, Dijkstra's algorithm solves the *single-source shortest path problem* on graphs with nonnegatively weighted edges; see ([CLR90] pages 527–532) for example. The version in Table 3.1 solves a variation of this problem; it finds the least edge-weight path from any vertex in S to all vertices in V. A well-known way of solving this multisource variation, without modifying Dijkstra's algorithm, is to modify the input graph by adding a new source vertex that is incident, via zero-weight edges, to all vertices in S. We will usually not have the luxury of effecting such modifications to our unit disk graphs when they are represented by sets of points in the plane. Therefore, Dijkstra's algorithm in Table 3.1 simulates the new source vertex and the $|S|$ new edges by "priming" the queue with the source vertices in S.

Dijkstra's algorithm uses a *priority queue* Q, a data structure that supports the following operations.

Q.**construct**(V,d) builds the priority queue containing the set V with keys d.

Q.**not-empty**() is true if Q does not contain any elements.

Q.**extract-min**() removes and returns the highest priority item, i.e., the one with the lowest key d_v, from the queue.

Table 3.1: **Algorithm:** DIJKSTRA(G,S) [Dijkstra's Algorithm]

Input: A nonnegatively weighted graph $G = (V, E, w)$ and a set of source vertices $S \subseteq V$.
Output: A shortest path forest, given by the parent array π, and the distance from S to every vertex, given by d.

```
for each vertex u ∈ V
    do if u ∈ S   /* Prime the queue with S. */
          then d_u ← 0   /* i.e., simulate w(s, u) = 0 */
          else  d_u ← ∞
       π_u ← NIL
Q.construct( V, d )
while Q.not-empty()
    do u ← Q.extract-min()
       L ← {v : (u, v) ∈ E}
       for each vertex v ∈ L
           do if d_v > d_u + w(u, v)
                 then d_v ← d_u + w(u, v)
                      Q.decrease-key(v, d_v)
                      π_v ← u
return (π, d)
```

Q.**decrease-key**(v,d_v) increases the priority of v by lowering its key to d_v.

Dijkstra's algorithm finds least edge-weight paths in $O((V+E)\log V)$ time when its priority queue is implemented as a binary heap [CLR90]. Note that this performance can be improved to $O(V \log V + E)$ time using Fibonacci heaps [CLR90], but the algorithms presented in this section do not need to do so since they attain their efficiency in other ways. Before modifying the algorithm any further, let us prove[1] that the key of any vertex removed from the queue (at Step 8) is no more than that of any subsequently removed vertex.

Lemma 3.9 (The Monotone Lemma) *The vertex keys form a nondecreasing sequence in the order that Dijkstra's algorithm removes them from the queue.*

Proof: We need only prove that Dijkstra's algorithm maintains the invariant that no vertex in the queue has a key less than that of an already removed vertex. This invariant holds trivially after the algorithm constructs the queue. Thereafter, the algorithm changes the priority of a vertex v only at Step 13, by calling Q.decrease-key(v, d_v). At this stage, vertex u has just been removed from the queue, and $d_v \leftarrow d_u + w(u,v)$. Therefore $d_v \geq d_u$, since all weights are nonnegative. By the invariant, d_u is no less than the key of any other removed vertex. It follows that d_v also is not less than that of any removed vertex. ∎

Vertex Weights

We can further modify Dijkstra's algorithm to solve the least *vertex-weight* path problem. Here, the weight of a path is the sum of the weights of its vertices. The vertex weights must be nonnegative, as was also true for the edge weights in the previous algorithm.

[1]This fact is probably well-known. Nevertheless, I was unable to find it stated in a standard reference, so I have chosen to prove it here for completeness.

Again, the usual way of solving this variation is to modify the graph, not the algorithm. To simulate the weight of a vertex with edge weights, set the weight of every incoming edge to the vertex weight. Since we wish to avoid explicit edge weights, we must change the algorithm by changing every occurrence of $w(u, v)$ to $w(v)$. In particular, we must change the already-simulated $w(s, u)$ in Step 3 from the constant 0 to $w(u)$ for all $u \in S$. Also, in Steps 11 and 12, we must change the edge-weight term $w(u, v)$ to the vertex-weight term $w(v)$. The modified algorithm VERTEX-DIJKSTRA is shown in Table 3.2.

Table 3.2: **Algorithm:** VERTEX-DIJKSTRA(G,S) [Dijkstra's algorithm for vertex weights]

Input: A nonnegatively vertex-weighted graph $G = (V, E, w)$ and a set of sources $S \subseteq V$.
Output: A least-weight path forest, given by the parent array π, and the weight from S to every vertex, given by d.

```
1   for each vertex u ∈ V
2       do if u ∈ S /* Prime the queue with S. */
3             then d_u ← w(u) /* i.e., simulate w(s, u) = w(u) */
4             else d_u ← ∞
5          π_u ← NIL
6   Q.construct(V, d)
7   while Q.not-empty()
8       do u ← Q.extract-min()
9          L ← {v : (u, v) ∈ E}
10         for each vertex v ∈ L
11             do if d_v > d_u + w(v)
12                   then d_v ← d_u + w(v)
13                        Q.decrease-key(v, d_v)
14                        π_v ← u
15  return (π, d)
```

The running time has not changed; it is still $O((V + E) \log V)$. We can do better, though, because algorithm VERTEX-DIJKSTRA changes the priority of each vertex exactly once, when the algorithm visits it for the first time. This is shown by the following lemma. Note that the lemma would not be true for Algorithm DIJKSTRA, that is, if the algorithm uses arbitrary edge weights.

Lemma 3.10 *Algorithm VERTEX-DIJKSTRA determines the final priority of a nonsource vertex the first time it removes an adjacent vertex from the queue.*

Proof: Let v be a nonsource vertex, and let u be the first adjacent vertex removed from the queue (at Step 8). Then $d_v \leftarrow d_u + w(v)$ at Step 12, and d_v either remains infinite if d_u is infinite, or attains some finite value if d_u is finite. Now, let u' be any subsequently removed vertex adjacent to v. By the Monotone Lemma, $d_u \leq d_{u'}$ when the algorithm removes u' from the queue (at Step 8). Therefore, $d_v = d_u + w(v) \leq d_{u'} + w(v)$, so that the test at Step 11 fails, and Step 13 does not get called to change the priority of v. ∎

This lemma implies that Algorithm VERTEX-DIJKSTRA, when recalculating priorities, need not examine all vertices adjacent to a removed vertex. Rather, it need only examine those nonsource vertices that are not adjacent to any previously removed vertices. To facilitate this, we can modify the algorithm to *mark* source vertices at Step 3.5 and adjacent nonsource vertices at Step 9.5. The new algorithm must also compute the set of adjacent unmarked vertices L accordingly. The complete algorithm MIN-PATH is shown in Table 3.3.

Lemma 3.11 *Algorithm MIN-PATH finds a lightest path forest in* $O(V \log V + E)$ *time.*

Proof: The correctness of the algorithm follows from the correctness of Dijkstra's algorithm and the preceding discussion. Using a binary heap priority queue, Q.construct() takes $O(V)$ time, Q.extract-min() takes $O(\log V)$ time, and Q.decrease-key() takes $O(\log V)$ time (see [CLR90] pages 140–152). Vertex operations therefore take $O(V \log V)$ time

Table 3.3: **Algorithm:** MIN-PATH(G,S) [Least Vertex-Weight Path]

Input: A nonnegatively vertex-weighted graph $G = (V, E, w)$ and a set of sources $S \subseteq V$.
Output: A lightest path forest, given by the parent array π, and the weight of the lightest path from S to every vertex, given by d.

for each vertex $u \in V$
  do if $u \in S$
    then $d_u \leftarrow w(u)$
      MARK(u)
    else $d_u \leftarrow \infty$
   $\pi_u \leftarrow$ NIL
Q.construct(V, d)
while Q.not-empty()
  do $u \leftarrow Q$.extract-min()
   $L \leftarrow \{v : v$ is not marked and $(u, v) \in E\}$
   MARK(L)
   for each vertex $v \in L$
    do if $d_v > d_u + w(v)$
      then $d_v \leftarrow d_u + w(v)$
       Q.decrease-key(v, d_v)
       $\pi_v \leftarrow u$
return (π, d)

since the algorithm removes each vertex from the queue exactly once, marks it exactly once, and therefore decreases its key at most once. The remaining step is Step 9, which examines every edge twice in a straightforward implementation. This takes $O(E)$ time and the lemma follows. ∎

Exploiting Unit Disk Graphs

Now that we have isolated the dependence of Algorithm MIN-PATH on the edges of the graph, we can eliminate this dependence for unit disk graphs. We will do this by efficiently implementing vertex marking, as well as Step 9 in Algorithm MIN-PATH, for unit disk graphs represented as a set of points in the plane. We will follow the theoretical framework proposed by Imai and Asano [IA86], who solve depth first search and breadth first search in $O(V \log V)$ time for another class of geometrically represented graphs. In the process, we will also derive depth first and breadth first search algorithms that run in $O(V \log V)$ time on unit disk graphs. We will require two basic operations: LIST1(v) and MARK(v). The MARK(v) operation marks a vertex v, as in Algorithm MIN-PATH. The operation LIST1(v) returns an unmarked vertex adjacent to v if one exists, and returns NIL otherwise. We can therefore implement Step 9 (which includes Step 9.5) in the MIN-PATH algorithm by calling Algorithm LIST-DELETE in Table 3.4. The following property of LIST-DELETE is straightforward.

Property 3.12 ([IA86]) *Every sequence of $O(V)$ calls to LIST-DELETE makes $O(V)$ calls to LIST1 and to MARK.*

This, together with our previous discussion of Algorithm MIN-PATH, proves the following lemma.

Table 3.4: **Algorithm:** LIST-DELETE(G, u) [Return and mark (delete) all unmarked vertices adjacent to u in G]

Input: A graph $G = (V, E)$ and a vertex $u \in V$
Output: $L = \{v : v \text{ is not marked and } (u, v) \in E\}$
Side Effect: All vertices in L are marked.

$v \leftarrow$ LIST1(u)
$L \leftarrow \emptyset$
while $v \neq$ NIL
  do $L \leftarrow L + \{v\}$
    MARK(v)
    $v \leftarrow$ LIST1(u)
return L

Lemma 3.13 *Suppose any sequence of $O(V)$ calls to LIST1 and MARK can be executed, on-line, in $g_t(V, E)$ $(= \Omega(V))$ time and $g_s(V, E)$ $(= \Omega(V))$ space. Then Algorithm MIN-PATH finds a lightest path forest in $O(g_t(V, E) + V \log V)$ time and $O(g_s(V, E))$ space.*

Proof: As in the proof of Theorem 3.11, Q.construct() takes $O(V)$ time, Q.extract-min() takes $O(\log V)$ time, and Q.decrease-key() takes $O(\log V)$ time using a binary heap priority queue. Vertex operations therefore take $O(g_t(V, E) + V \log V)$ time since the algorithm removes each vertex from the queue exactly once, marks it exactly once, and therefore decreases its key at most once. The remaining step is Step 9, which the algorithm executes exactly once for each of $|V|$ vertices on the queue. This therefore takes $O(g_t(V, E))$ time by Property 3.12, and the lemma follows. ∎

Suppose a unit disk graph $G = (V, E)$ is represented by a realization $f : V \to \mathbf{R}^2$. Then we can implement a call to LIST1(v) with a *disk query*, that is, by finding a point in $f(V)$ that lies within the unit radius disk centered at $f(v)$. We can implement a call to

MARK(v) by simply deleting v from V. We will see how to implement these operations efficiently in Section 3.2.2. In particular, we will prove the following theorem.

Theorem 3.17 (Section 3.2.2) *Given a finite set of points $f(V)$ in the plane, a sequence of $O(V)$ disk queries and point deletions execute in $O(V \log V)$ time, given $O(V \log V)$ preprocessing time with $O(V)$ space.*

Since LIST1 and MARK are effectively equivalent to disk query and site deletion respectively, we immediately have the following corollary.

Corollary 3.13.1 *A sequence of $O(V)$ calls to LIST1 and MARK on unit disk graphs can be executed in $O(V \log V)$ time and $O(V)$ space.*

Lemma 3.13 and Corollary 3.13.1 therefore prove the main result of this section, as stated by the following theorem.

Theorem 3.14 *Given a representation of a unit disk graph by its realization (V, f), a least vertex-weight path can be found in $O(V \log V)$ time and $O(V)$ space.*

In addition to finding least weight paths, efficiently, we can also perform an efficient depth first search on a unit disk graph, given a realization. Imai and Asano [IA86] show that depth first search can be executed quickly if LIST1 and MARK can be executed quickly. The following theorem makes this more precise.

Theorem 3.15 ([IA86]) *Suppose a sequence of $O(V)$ calls to LIST1 and MARK can be executed, on-line, in $g_t(V, E)$ $(= \Omega(V))$ time and $g_s(V, E)$ $(= \Omega(V))$ space. Depth first search can be executed in $O(g_t(V, E))$ time and $O(g_s(V, E))$ space.*

Theorem 3.16 below applies Theorem 3.15 to unit disk graphs. We will use Theorem 3.16 in Section 5.1 to determine if unit disk graphs are connected in $O(V \log V)$ time.

Theorem 3.16 *Depth first search on unit disk graphs can be executed in $O(V \log V)$ time and $O(V)$ space, given a geometric representation.*

Proof: The theorem follows from Theorem 3.15 and Corollary 3.13.1. ▮

3.2.2 Semi-Dynamic Single-Point Circular-Range Queries

We are now ready to discuss disk queries in more detail. Given a finite set S of *sites* (points in the plane), a *disk query* asks for just one site inside a (closed) query disk with unit radius, or a report that no such site exists. Equivalently, it asks for a site within unit distance of a *query point*, that is, the center of the query disk. In addition, we want to delete sites from the set. This section presents an algorithm that executes a disk query operation in $O(\log S)$ time and deletes a site in $O(\log S)$ amortized time, given $O(S \log S)$ preprocessing time with $O(S)$ space.

The disk query problem resembles circular range searching—report all sites in a query disk [PS85]. Another related problem is nearest neighbour search—report the site closest to the query—which can be solved with Voronoi diagrams [PS85]. These results are not immediately useful for our needs, since they do not support deletions efficiently. Note, however, that the algorithms developed in this section are not fully dynamic in that they do not support insertion.

The Data Structure

At the coarsest level, our disk-query data structure is a square grid in the plane. The $|S|$ sites fall into the cells of this grid, and the contents of each cell are organized in a data structure. At a finer level, the data structure explicitly represents only those cells that contain sites, $|S|$ cells at most. More concretely, construct the data structure by conceptually partitioning the plane into square cells with unit diagonals, that is, with

$\sqrt{1/2}$ sides. To avoid ambiguous site placement, assume that each cell is closed on the left and bottom, and open on the right and top. Label a cell with the coordinates of its lower left corner. We can determine the label of the cell that contains a given site in constant time since our model of computation includes the floor function. The data structure is a balanced tree (ordered lexicographically by label) of those cells that contain sites. In this way we can determine in which cell in the tree a site lies, or determine that the site is not in any cell in the tree, in $O(\log S)$ time. We can also insert a new cell in $O(\log S)$ time. We can therefore create the complete tree of cells in $O(S \log S)$ time as follows. For each site v, determine if the cell containing v is already in the tree. If not, create a new cell, insert it into the tree, and associate a linked list (containing just v) with the cell. Otherwise, just append v to the end of the list associated with the cell in constant time. Each cell must now organize its list of sites into its own data structure.

Searching Cells

Before discussing how the sites within a cell are organized, let us first examine the structure of a query with respect to these cells. Note that the cells covered by any query disk come from a set of 21 cells as shown in Figure 3.5. We can therefore answer the query by looking in at most 21 cells.

Consider the cell in which the center of the query disk lies. This cell always lies completely within the query disk. Therefore, if it is in the tree, any site within it satisfies the query.

The other covered cells lie either strictly to the left, to the right, above, or below the query point. Suppose first that the cell lies strictly below the query point $q = (x_q, y_q)$; the other three cases are handled symmetrically. We can therefore assume that our query disk is in fact a query "test tube", that is, the union of the query disk and the infinite

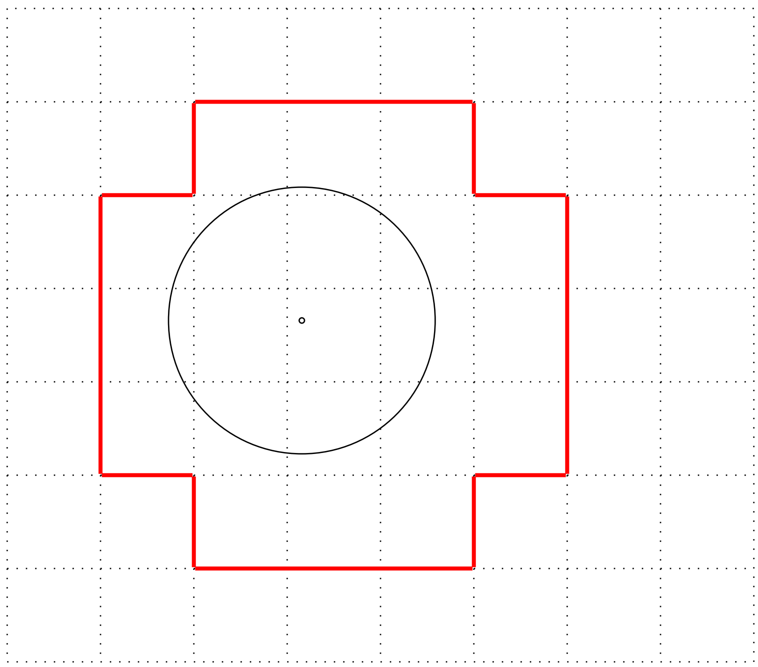


Figure 3.5: Any unit-radius query disk covers at most 21 cells.

rectangle $[x_q - 1, x_q + 1] \times [y_q, \infty)$. The *center* of the test tube is still defined to be (x_q, y_q).

The following definitions and observations are from Appendix A. A tube T *contains* a subset $P \subseteq S$ of sites if $P \subseteq T$, and it is *empty* if it does not contain any sites. A tube and a site on its boundary are said to be *incident*. A tube is said to be *supporting (with respect to S)* if it is incident to exactly one site and is otherwise empty. A site in S is said to be a *test tube maximal site with respect to S*, or just a *maximal*, if it is incident to some supporting tube. By the following lemma, if we wish to report a site in a test tube, it suffices to examine only maximal sites.

Lemma A.2 (Appendix A) *A test tube contains a site if and only if it contains a maximal site.*

Binary Searching Maxima

Order the maxima from left to right. Say that two maxima are *neighbouring* if they are adjacent in the order. Then binary search can be used to report a maximal in a query test tube. The following lemmas provide the details. The proofs are presented in the appendix.

Lemma A.3 (Appendix A) *If a site p is incident to a test tube that contains a site to the left of p and a site to the right of p, then p is not maximal.*

Lemma A.5 (Appendix A) *Let p be a maximal and let T_p be any incident test tube. If T_p does not contain the right neighbour of p, then T_p does not contain* any *sites to the right of p. Similarly, if T_p does not contain the left neighbour of p, then T_p does not contain any sites to the left of p.*

We are now ready to search for a maximal in a test tube. Let p be an arbitrary maximal and let T be the query test tube. If T contains p, then we are done. If p lies strictly left of T, then any sites in T must lie to the right of p. Similarly, if p lies strictly to the right of T, then any sites in T must lie to the left of p. Otherwise, if p lies neither strictly to the left nor strictly to the right of T, then we can lower T onto p to get test tube T'. If T' does not contain any sites, then neither does T (since T is strictly contained by T'). By Lemma A.3, at most one neighbour of p lies in T'. By Lemma A.5, if no neighbour lies in T', no sites at all lie in T' and we are done. Also by Lemma A.5, if a neighbour does lie in T', then any other sites in T' (and therefore any sites in T) must lie on this neighbour's side of p.

Any data structure that supports binary search can therefore determine, in $O(\log S)$ time, if one of the (at most $|S|$) maxima lies in a given test tube T. Such a data structure is developed in Appendix A; the *test tube data structure* represents the test tube maxima of the set of sites S. The following two theorems from the appendix describe the performance of this data structure.

Theorem A.11 (Appendix A) *The test tube data structure $(B(S), D(S))$ of sites S can be built in $O(S \log S)$ time and $O(S)$ space.*

Theorem A.17 (Appendix A) *$O(S)$ sites can be deleted from the test tube data structure $(B(S), D(S))$ of sites S in $O(S \log S)$ time.*

Since any cell could be either to the left, right, above, or below a query point, each cell must maintain four sets of maxima. We can derive the other three directions for cells symmetrically, by rotating the coordinate system in 90 degree increments. This completes the disk query data structure.

A Summary of Algorithms

In summary, the data structure requires the following three algorithms. The *preprocessing algorithm* builds the data structure by constructing the tree of cells in $O(S \log S)$ time. Then, for each cell b, it builds four sets of maxima from its n_b sites in $O(n_b \log n_b)$ time. Since $\sum_b n_b = |S|$, and $\log b \leq \log |S|$, the sum of the $O(n_b \log n_b)$ costs over all nonempty cells is $O(S \log S)$.

To find a site in a query disk, the *query algorithm* must examine at most 21 cells. The locations of these cells are all closely related to the location of the cell containing the query point, so the query algorithm can find each each such cell in $O(\log S)$ time. The query algorithm can then extract a suitable site in constant time from the center cell, or in $O(\log n_b)$ time from any of the others.

To delete a site, the *deletion algorithm* finds the cell that contains the site in the tree in $O(\log S)$ time. It then deletes the site from all four maxima lists in $O(\log n_b) = O(\log S)$ amortized time. This proves the following theorem.

Theorem 3.17 *Let S be a set of sites in the plane. A sequence of $O(S)$ disk queries and site deletions execute in $O(S \log S)$ time, given $O(S \log S)$ preprocessing time and $O(S)$ space.*

3.2.3 Building an Adjacency List from a Model

Recall from Definition 1.2 that the *unit disk graph $G_\delta(P)$ generated by a set V of points and a real threshold δ* is the graph (V, E) where $E = \{(p, q) : \|p - q\| \leq \delta\}$. How does $G_\delta(P)$ change as δ changes? In particular, how does the number of edges change? Surprisingly, the number of edges generated by two different distance units are linearly related (Theorem 3.18 below).

We saw how a geometric realization of a unit disk graph can be used to implicitly represent the edges, and consequently to design algorithms whose computational complexity does not depend on the number of edges. What can we do if we have only the geometric representation of the graph, and we wish to compute its edges? One solution would be to simply look at every pair of vertices, and check if the corresponding points are within unit distance of one another. This method would take $\Theta(V^2)$ time, which is optimal for dense graphs.

We can do better if our graphs are sparse. Theorem 3.18 makes it possible to report efficiently the edges of a generated unit disk graph $G_\delta(V) = (V, E_\delta)$ in $O(V \log V + E_\delta)$ time. This section demonstrates this property by presenting and analyzing two different algorithms. Given a set V of points in the plane, these algorithms report the edges E_δ in the unit disk graph $G_\delta(V, E_\delta)$ for all distance thresholds δ. Although these two algorithms are correct for different reasons, we will see that their efficiency is due to Theorem 3.18. This reporting problem is also known in the literature [BSW77, DD90, LS94] as the *fixed-radius all-nearest neighbours problem*: given a set $P \in \mathbf{R}^k$ of points and a fixed radius δ, report all pairs of points within δ of each other.

Theorem 3.18 *Let $\delta < \Delta$ be two nonnegative distance thresholds. If $G_\delta(V) = (V, E_\delta)$ and $G_\Delta(V) = (V, E_\Delta)$ are unit distance graphs generated by the same set V of sites, then $|E_\Delta| = O(V + E_\delta)$.*

Proof: This proof generalizes a demonstration by Dickerson and Drysdale [DD90] showing (effectively) that $|E_2| = O(V + E_1)$. Let $N_\Delta(v)$ denote the closed neighbourhood of v in G_Δ. That is, $N_\Delta(v)$ is the set of sites in P within Δ units of $f(v)$. To simplify the presentation, let us assume that all graphs have loops, that is, that $(v, v) \in E(G)$ for every $v \in V(G)$. Let us also assume that each undirected edge is really two undirected edges. Then

$$E_\Delta = \bigcup_{v \in V} \{(u, v) : u \in N_\Delta(v)\}$$

and

$$|E_\Delta| = \sum_{v \in V} |N_\Delta(v)|. \tag{3.1}$$

The rest of this proof shows that $\sum_{v \in V} |N_\Delta(v)| = O(E_\delta)$ by deriving a lower bound for $|E_\delta|$. The basic idea is to partition the plane into regions with diameter at most δ. Any pair of sites in such a region will generate an edge in E_δ. We will then associate, with each site v, a region containing at least a constant fraction of the sites in $N_\Delta(v)$. Each such region therefore contributes roughly $|N_\Delta(v)|^2$ edges to E_δ. Although a region might be associated with more than one site, it will not be associated with more sites than a constant multiple of $|N_\Delta(v)|$. The (disjoint) sum of edges in these associated regions is therefore at least roughly

$$\sum_{v \in V} (|N_\Delta(v)|^2 / |N_\Delta(v)|) = \sum_{v \in V} |N_\Delta(v)|.$$

More concretely (and more carefully), overlay the plane with a square grid whose cell diagonals are δ units long, that is, whose sides are $\delta/\sqrt{2}$ units long. Let the cells be closed on the left and bottom, and open on the right and top. These cells clearly partition the plane and have diameter at most δ, as outlined in the previous paragraph.

For any vertex v, the neighbourhood $N_\Delta(v)$ comes from at most a constant number k_1 of these grid cells. The exact value of this constant is not crucial, but it is nevertheless easy to exhibit a value that will suffice. For example, there are at most

$$k = \lfloor \frac{\Delta}{\delta/\sqrt{2}} \rfloor = \lfloor \frac{\Delta}{\delta}\sqrt{2} \rfloor \tag{3.2}$$

columns strictly between v and the leftmost site in $N_\Delta(v)$, since each cell has width $\delta/\sqrt{2}$. Therefore, counting the leftmost, rightmost, and center columns, the sites in $N_\Delta(v)$ come from at most $3 + 2k$ columns. The same argument holds for rows, so $N_\Delta(v)$ touches at most $k_1 = (3 + 2k)^2$ cells.

Let the *maximum cell* C_v be the cell that contains the most sites from $N_\Delta(v)$ (break ties arbitrarily). The Pigeonhole principle implies that the maximum cell C_v contains at least $|N_\Delta(v)|/k_1$ sites from $N_\Delta(v)$. That is,

$$|C_v| \geq \frac{|N_\Delta(v)|}{k_1} \tag{3.3}$$

where $|C_v|$ is the number of sites in C_v (note that not all of these $|C_v|$ sites need belong to $N_\Delta(v)$). As promised, every pair of sites in C_v generates an edge in E_δ since every such pair is separated by at most δ units. Does this mean that $|E_\delta| \geq \sum_{v \in V} |C_v|^2$? No, because cell C_v may be counted more than once in this way if it is the maximum cell for more than one site, that is, if $C_v = C_u$ for some site $u \neq v$.

Fortunately, there is a constant k_2 such that no cell C_v is counted more than $k_2|C_v|$ times. Again, the exact value of k_2 is not crucial, but we can easily derive a suitable one. Suppose $C_u = C_v$ for some site u. Then there are again at most k columns (see Equation 3.2) strictly between u and C_v. By the same argument as above, u must come from one of $k_2 = (3 + k)^2$ cells. But if the cell C^* containing u contains more than $|C_v|$ sites, then C_v would could not be the maximal cell for u, since C^* contains only sites from $N_\Delta(u)$. Therefore u must be one of at most $k_2|C_v|$ sites.

A lower bound for the number of edges in G_δ is, therefore,

$$\begin{aligned}|E_\delta| &\geq \sum_{v\in V} \frac{\text{number of pairs in } C_v}{|\{u : C_u = C_v\}|} \\ &\geq \sum_{v\in V} \frac{|C_v|^2}{k_2|C_v|} \\ &= \frac{1}{k_2}\sum_{v\in V}|C_v| \end{aligned} \tag{3.4}$$

Substituting Inequality 3.3 into Inequality 3.4 yields

$$\begin{aligned}|E_\delta| &\geq \frac{1}{k_2}\sum_{v\in V}\frac{|N_\Delta(v)|}{k_1} \\ &= \frac{1}{k_1k_2}\sum_{v\in V}|N_\Delta(v)|\end{aligned}$$

That is,

$$\sum_{v\in V}|N_\Delta(v)| \leq k_1k_2|E_\delta| \tag{3.5}$$

Finally, substituting Inequality 3.5 into Inequality 3.1,

$$\begin{aligned}|E_\Delta| &\leq \sum_{v\in V}|N_\Delta(v)| \\ &\leq k_1k_2|E_\delta|.\end{aligned}$$

Finally, if E'_δ and E'_Δ are versions of E_δ and E_Δ without loops and *not* directed, then $|E'_\Delta| = O(E_\Delta) = O(E_\delta) = O(V + E'_\delta)$, and the theorem follows.

Note that we cannot do without the $O(V)$ term in general. For example, if $V = \{(x, 0) : x = 0, 1, 2, \ldots, n\}$, $\delta = 0.5$, and $\Delta = 2\delta$, then $G_\delta(V)$ has no edges at all, while $G_\Delta(V)$ has n edges. However, this is only true if $G_\delta(V)$ is not connected, as shown by the following corollary. ∎

Corollary 3.18.1 *Let $\delta < \Delta$ be two nonnegative distance thresholds. If $G_\delta(V) = (V, E_\delta)$ and $G_\Delta(V) = (V, E_\Delta)$ are unit distance graphs generated by the same set V of sites, and $G_\delta(V)$ is connected, then $|E_\Delta| = O(E_\delta)$.*

Proof: The corollary follows from the theorem since $|V| = O(E_\delta)$ for connected graphs $G_\delta = (V, E_\delta)$. ∎

A Plane-Sweep Algorithm

To simplify the notation for Algorithm ADJ-PLANE-SWEEP (Table 3.5), assume a unit distance threshold $\delta = 1$. Algorithm ADJ-PLANE-SWEEP maintains three indices p, q, and r; and two sets L and Bar of points. The set L is a one-dimensional array of sites, indexed from 1 to $|V|$, and p, q, and r are indices into L; for convenience, write $(x_p, y_p) = L_p$. Array L stores a lexicographically sorted list of the sites in V. That is, $p < q$ implies $x_p < x_q$, or $x_p = x_q$ and $y_p < y_q$. The array L supports two operations: L.build(V), which constructs the lexicographically sorted list; and L_i, which returns the site indexed by i. For each $p \in V$, define the set

$$Bar_p = \{q : q \in V,\ q < p,\ \text{and}\ x_p - 1 \leq x_q\}.$$

ADJ-PLANE-SWEEP reuses storage by dropping the subscript, thereby representing *each* such set with the single data structure Bar. See Figure 3.6 for a typical event in the plane-sweep. Algorithm ADJ-PLANE-SWEEP examines each site in L from left to right using the index p. It maintains the invariant that indices r and p delimit the contents of the set Bar. To do so, it increases r until $x_p - 1 \leq x_r$. In the process, it removes all sites that have dropped out of the bar.

Correctness Let $G(V) = (V, E)$ be the unit disk graph generated by V and suppose that $(L_p, L_q) \in E$. By definition, $\|L_p - L_q\| \leq 1$. Suppose, without loss of generality, that $q < p$. Therefore q will be in the Bar when the algorithm encounters p since $|x_p - x_q| \leq 1$. The algorithm therefore correctly reports edge (L_p, L_q) at Step 9 when it examines p. Furthermore, the test in Step 8 ensures that it reports only edges in $G(V)$.

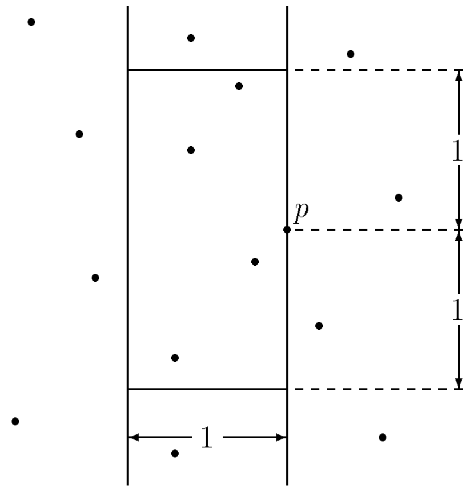


Figure 3.6: Algorithm ADJ-PLANE-SWEEP examines site p

Table 3.5: **Algorithm:** ADJ-PLANE-SWEEP(V) [Construct the list of edges for the unit disk graph $G(V)$ generated by V by sweeping the plane.]

Input: A set V of sites.
Output: The edges E of the unit disk graph $G(V) = (V, E)$ generated by V.

L.build(V)
$E \leftarrow \emptyset$
$p \leftarrow r \leftarrow 1$
while $p \leq |V|$ /* examine L_p */
    do while $r < p$ and $x_r < x_p - 1$ /* Bring Bar up to date */
        do $Bar \leftarrow Bar \setminus \{r\}$
            $r \leftarrow r + 1$
    for each site $q \in Bar$ such that $y_q \in [y_p - 1, y_p + 1]$
        do if $\|L_p - L_q\| \leq 1$
            then $E \leftarrow E \cup \{(L_p, L_q)\}$
    $Bar \leftarrow Bar \cup \{p\}$
    $p \leftarrow p + 1$
return E

Complexity This algorithm can be implemented to run in $O(V \log V + E)$ time. The preprocessing operation L.build(V) just sorts the sites in $O(V \log V)$ time.

Since the algorithm inserts each site into the *Bar* exactly once and removes each site at most once, it makes $O(V)$ insertions and deletions. The implementation represents the *Bar* as a balanced tree, a red-black tree for example [Sed83], sorted by y-coordinate. Inserting (Step 10) or deleting (Step 5) a site therefore takes $O(\log V)$ time for a total of $O(V \log V)$ time. Balanced trees support the extraction of sites in Step 7 in $O(\log V + I_p)$ time, where I_p is the number of sites in the unit interval about y_p [Sed83]. Examining all V intervals therefore takes

$$O(V \log V + \sum_{p=1}^{V} I_p)$$

time.

The sites examined by Step 7 are all within $\sqrt{2}$ of p (see Figure 3.6 again). Therefore $\sum_{p=1}^{|V|} I_p \leq |E_{\sqrt{2}}|$ where $G_{\sqrt{2}}(V) = (V, E_{\sqrt{2}})$. By Theorem 3.18, it follows that:

$$\sum_{p=1}^{|V|} I_p = O(V + E).$$

The entire algorithm therefore takes $O(V \log V) + O(V + E) = O(V \log V + E)$ time. We have established the following theorem.

Theorem 3.19 *Algorithm ADJ-PLANE-SWEEP returns the edges of the unit disk graph $G(V) = (V, E)$ generated by a set V of points in $O(V \log V + E)$ time.*

A Delaunay Triangulation Algorithm

The next algorithm is due to Dickerson and Drysdale [DD90]. It has the advantage that it allows a preprocessing stage, which is time-critical, and which is independent of the distance threshold δ. This preprocessing stage takes $O(V \log V)$ time. The algorithm can then report the edges in $G_\delta(V)$ in $O(V + E)$ additional time. Briefly, the algorithm performs a depth-first search on the *thresholded Delaunay triangulation* TDT(V)—the Delaunay triangulation of V less any edges that are longer than δ—from every vertex $p \in V$. The algorithm stops the current branch whenever it visits a vertex that is more than δ units from p. The algorithm reports an edge from p at all other visited vertices. Table 3.6 provides more details.

Table 3.6: **Algorithm:** ADJ-DELAUNAY(V, δ) [Construct $G_\delta(V)$ by searching a thresholded Delaunay triangulation]

Input: A set V of sites and a distance threshold δ.
Output: The edges E_δ of the unit disk graph $G_\delta(V) = (V, E_\delta)$ generated by V.

Construct the Delaunay triangulation $DT(V)$ of V.
TDT(V) $\leftarrow DT(V) \setminus \{(p, q) : \|p - q\| > \delta \}$
$E_\delta \leftarrow \emptyset$
for every vertex $p \in V$ /* examine p */
  do depth first search TDT(V) from p, but stop each branch when
    the current site is farther than δ from p.
    for each visited vertex q
      do if $\|p - q\| \leq \delta$
        then $E_\delta \leftarrow E_\delta \cup \{(p, q)\}$
return E_δ

Correctness Let (p, q) be an edge of the unit disk graph generated by V. Then there is a path from p to q, in the Delaunay triangulation of V, such that every site on the path is within δ units of p. To see this, consider the line segment between p and q. This line segment intersects several adjacent cells in the Voronoi diagram of the sites V. The sequence $(p = O_1, O_2, \ldots, O_k = q)$ of sites corresponding to these Voronoi cells is therefore a path in the Delaunay triangulation by the duality of the Delaunay triangulation and the Voronoi diagram (see Section 2.4.1). Now consider any site (Voronoi center) O_i in this sequence, and let x be any point on the part of the segment intersecting the Voronoi

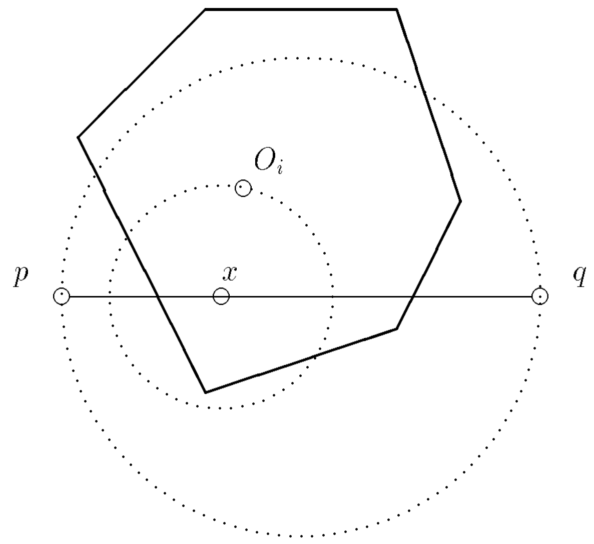


Figure 3.7: The disk through p and q contains Voronoi center O_i

cell corresponding to O_i (see Figure 3.7). By the definition of Voronoi diagram, x is closer to O_i than to any other site in V. It follows that the circle about x and through O_i (that is, of radius $\|O_i - x\|$) does not contain any other site in V. In particular, it does not contain p or q (unless O_i equals p or q respectively). The circle with diametric points p and q therefore contains the circle about x and so contains O_i also. Since in addition $\|p - q\| \leq \delta$, the site O_i is within δ units of p and q and of all other sites O_j in the path. That is, $(O_1, O_2, \ldots, O_k)$ is a path from p to q in the thresholded Delaunay triangulation. Consequently, Step 7 will report the edge (p, q) when it examines p.

Complexity Let $G^p = (V^p, E^p)$ be the connected component (of the thresholded Delaunay triangulation $TDT(V)$) that contains p. Suppose that Step 4 visits a vertex q so that $q \in V^p$. Since Step 1 thresholds the Delaunay triangulation, q must be within δ of its parent in the depth first search. This parent must be within δ of p since Step 4 cuts off depth first search when the distance from p to the current site exceeds δ. Therefore, q is within 2δ of p by the triangle inequality. It follows that $(p,q) \in E^p_{2\delta}$ where $G_{2\delta}(V^p) = (V^p, E^p_{2\delta})$. Since G^p is connected, Corollary 3.18.1 applies to show that $E^p_{2\delta} = O(E^p)$. Since $\sum_{p \in V} |E^p| = |E|$, it follows that $\sum_{p \in V} |E^p_{2\delta}| = O(E)$ so that Step 6 executes $O(E)$ times.

The Delaunay triangulation can be constructed in $O(V \log V)$ time [PS85], and thresholded in $O(V)$ time (the number of edges of a Delaunay triangulation is linear in the number of sites). However, Volker Turau has discovered a way to implicitly threshold the Delaunay triangulation in the preprocessing step [Tur91], thereby saving the $O(V)$ thresholding cost and the $\Theta(V)$ cost of examining each vertex in Step 3. Represent the Delaunay triangulation as an adjacency list. List every vertex p in nondecreasing order of the length of the shortest edge incident with p. List every neighbour q of each vertex p in nondecreasing order of the length of edge (p,q). Then, for any given threshold δ, Step 3 can examine the vertices in order, stopping when it encounters a vertex with every incident edge longer than δ. In this way, it examines only vertices that are not isolated in G_δ (and at most one more). Similarly, Step 4 examines edges out of q in order, stopping when it encounters an edge longer than δ (charge the cost of detecting a long edge to visiting q).

The following theorem summarizes.

Theorem 3.20 **([DD90, Tur91])** *Given a set V of points in the plane, algorithm ADJ-DELAUNAY takes $O(V \log V)$ preprocessing time and, given a threshold δ, returns the edges of the unit disk graph $G_\delta(V) = (V,E)$ in $O(E)$ additional time.*

The discussion above implies that the connected components of a unit disk graph are equivalently the connected components of the underlying thresholded Delaunay triangulation. These connected components can therefore be extracted in linear time by depth first searching the thresholded Delaunay triangulation.

Theorem 3.21 *The connected components of a unit disk graph $G_\delta(V)$ can be found in $O(V \log V)$ time, given a set V of points in the plane.*

Other Work

There are other solutions to the fixed-radius all-nearest neighbour problem in the literature. Bentley, Stanat, and Williams [BSW77] have developed a solution that involves partitioning the plane into a grid of square cells, much like the *proof* of Theorem 3.18. They presented their algorithm in terms of the L_∞ metric in k-dimensional space, but it is not difficult to modify it to work with the L_2 metric in the plane. Again, Theorem 3.18 explains the computational complexity of their algorithm. However, the time required to access the cells containing sites in their algorithms is also a factor in the computational complexity. Bentley *et al.* mention three possible ways to implement this set of cells, leading to three different time bounds for the complete algorithm, as the following table summarizes.

Implementation of Occupied Cells	Time Complexity
hash table	$O(V + E)$ average case
balanced tree	$O(V \log V + E)$ worst case
two-dimensional array	$O(V + E)$ worst case

The two-dimensional array solution actually allocates space for all cells, occupied or not, that intersect the bounding rectangle of the sites. Since the $|S|$ sites may have an arbitrarily large range of coordinates, the two-dimensional array may require arbitrarily large space. The hash table or the balanced tree solutions, on the other hand, require only a linear amount of space.

Subsequent to (and independent of) the development of Algorithm ADJ-PLANE-SWEEP, Lenhof and Smid [LS92, LS94] described a similar algorithm that runs in

Problem	Unit Disk Performance Ratio	Arbitrary Graph Performance Ratio
chromatic number	3	$O(V(G)\frac{(\log\log V(G))^2}{(\log V(G))^3})$ [Hal90]
vertex cover	3/2	2 [GJ79]
independent set	3	not constant [ALM+92]
domination	5	$O(\log V)$ [Joh74]
independent domination	5	not constant [Irv91]
connected domination	10	?
total domination	10	?

Table 3.7: Performance ratios for approximation agorithms on unit disk graphs

$O(V \log V + E)$ time. Their approach is in effect a combination of our plane-sweep algorithm and the cell method described in the last paragraph. This combination makes it easier to adapt to higher dimensions than the pure plane-sweep algorithm. One attractive feature of their presentation is that they have recorded an animation of their algorithm on an easily-obtained videotape [LS94].

3.3 NP-Complete Problems Restricted to Unit Disk Graphs

Many well-known **NP**-complete problems on arbitrary graphs, in particular chromatic number, vertex cover, independent set, domination, independent domination, and connected domination [GJ79] remain **NP**-complete for unit disk graphs. Clark, Colbourn, and Johnson provide reductions for all of these problems in their recent paper [CCJ90]. Remarkably, all of these problems have efficient approximation algorithms. These algorithms, due to Marathe, Breu, Hunt, Ravi, and Rosenkrantz [MHR92, MBH+95], achieve the performance ratios in Table 3.7, which compares them with the best-known performance ratios for arbitrary graphs. Section 3.3.3 presents the details for the chromatic number problem.

Clark *et al.* notice that "For all of the problems[2] mentioned here, the complexities for unit disk graphs and for planar graphs agree." We saw in Chapter 1 that there are some tantalizing connections between planar graphs and unit disk graphs. Nevertheless, the observation by Clark *et al.* may be misleading in its generality. In fact, the complexities of these problems agree for unit disk graphs and for planar graphs with maximum degree 4.

Conjecture 3.22 *If a problem $\mathcal{P}$ is* **NP***-complete for planar graphs with maximum degree 4, then $\mathcal{P}$ is* **NP***-complete for unit disk graphs, too.*

Discussion For example, every grid graph is a planar graph with no degree exceeding 4. If a problem is **NP**-complete for grid graphs then it is clearly **NP**-complete for unit disk graphs, which include grid graphs as a special case. For example, the problem DOMINATING SET has this characteristic [CCJ90]. But there seem to be stronger connections. If a problem is **NP**-complete for planar graphs with maximum degree 4, then there is a "plan of attack" for proving that the problem is also **NP**-complete for unit disk graphs. The idea is to embed the planar graph instance in a grid graph, then simulate the embedded edges with a string of disks. The nature of the simulation naturally depends on the problem under consideration. This "plan of attack" is unfortunately not sufficiently well-defined to yield an easily applied theorem. Instead, the next subsection applies the idea to an illustrative example, namely INDEPENDENT SET. The **NP**-completeness proof for CHROMATIC NUMBER on unit disk graphs (Section 3.3.3) provides another example. Clark *et al.* [CCJ90], who provide an explicit reduction for VERTEX COVER rather than INDEPENDENT SET, provide even more examples.

[2]Besides these **NP**-complete problems, Clark *et al.* also consider the maximum clique problem, which is polynomial for both planar and unit disk graphs. See Section 3.3.2 for more details.

3.3.1 Independent Set

An *independent set*, also called a *stable set* in a graph is a set of vertices no two of which are adjacent. The maximum independent set problem is to find an independent set of greatest cardinality. This fundamental problem is very well known; the corresponding decision problem is given below, and is taken verbatim from Garey and Johnson's book [GJ79].

[GT20] **INDEPENDENT SET**

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.

QUESTION: Does G contain an independent set of size K or more, i.e., a subset $V' \subseteq V$ such that $|V'| \geq K$ and such that no two vertices in V' are joined by an edge in E?

Theorem 3.23 ([CCJ90]) *INDEPENDENT SET is* **NP**-*complete for unit disk graphs under the* L_1, L_2, *and* L_∞ *metrics.*

Proof: The problem is in **NP** since INDEPENDENT SET is **NP**-complete for arbitrary graphs [GJ79]. Let us reduce INDEPENDENT SET, which remains **NP**-complete [GJ79] for planar cubic graphs, to INDEPENDENT SET for unit disk graphs. To this end, let G_{pc} and k be an instance of INDEPENDENT SET on a planar cubic graph.

Begin by embedding the instance in a grid graph; this is done only to leave enough room between vertices and edges for the next step. Let Gd_{IJ} be the grid graph induced by the vertex set $\{(i, j) : 0 \leq i < I,\ 0 \leq j < J\}$. Valiant [Val81] shows how to embed, without crossovers, any planar graph of degree 4 (or less) with n vertices into $Gd_{3n,3n}$. His polynomial time method is simple: given any embedding of the graph in the plane, strip off one vertex at a time from the "perimeter" of the embedding. Embed these vertices in reverse order into the grid. The mth vertex is typically embedded as shown in Figure 3.8.

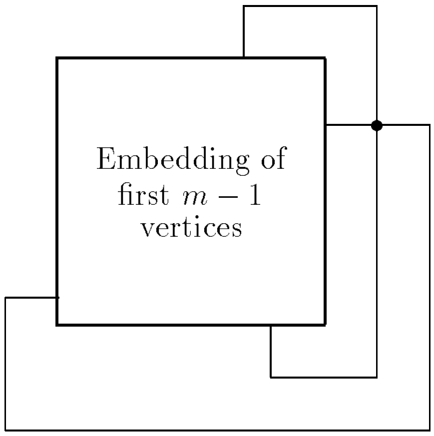


Figure 3.8: Embedding the mth vertex [after [Val81]].

Next, simulate the vertices of the planar graph with disks, and simulate the edges of the planar graph with a string of an even number of disks. We can use the grid embedding to make this concrete. All disks in the layout have diameter 1/3. Center the disks at the points P determined as follows. We must simulate three kinds of grid graph components: nodes, pseudo-nodes, and segments. Here, *nodes* are those grid graph vertices in the embedding that correspond to vertices of the planar cubic graph and *pseudo-nodes* are those that do not. *Segments* are grid graph edges in the embedding. Figure 3.9 shows how to simulate each of these components with disks. The figure shows two types of pseudo-nodes, those that bend and those that do not, even though they are simulated in the same way. In terms of coordinates, simulate

- the node (i, j) with a disk centered at (i, j),
- the segment $[(i, j), (i \pm 1, j)]$ with two disks, centered at $(i \pm \frac{1}{3}, j)$ and $(i \pm \frac{2}{3}, j)$,
- the segment $[(i, j), (i, j \pm 1)]$ with two disks, centered at $(i, j \pm \frac{1}{3})$ and $(i, j \pm \frac{2}{3})$,

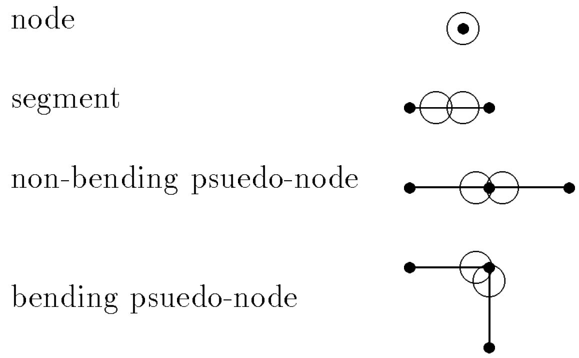


Figure 3.9: Components for simulating grid embeddings

- the pseudo-node at (i, j) by putting a disk at $(i \pm \frac{1}{6}, j)$ for each embedding fragment of the form $[(i, j), (i \pm 1, j)]$, and a disk at $(i, j \pm \frac{1}{6})$ for each embedding fragment of the form $[(i, j), (i, j \pm 1)]$.

Figure 3.10 simulates the embedding of a small graph (5 vertices, 6 edges). Note that every edge of the planar cubic graph is simulated with an *even* number of disks, as required.

Let n denote the number of nodes, p the number of pseudo-nodes, and s the number of segments in the embedding. The size of the simulation is polynomial in the size of the planar cubic graph since, by the properties of Valiant's embedding,

$$n + p + s \leq (3|V|)^2 + 2(3|V|^2) + 2(3|V| - 1)(3|V|) = O(V^2)$$

The planar cubic graph G_{pc} has an independent set I_{pc} of size k or more if and only if the unit disk graph $G_{ud} = (V, E)$ has an independent set I_{ud} of size $k' \geq k + p + s$. The following definitions and observation aid in seeing this. Say that a disk in the realization of G_{ud} is *selected* if it is in the independent set I_{ud}. Say that an edge in G_{pc} is *saturated* if one half of the disks that simulate it are selected. Equivalently, an edge is saturated if exactly one disk is selected from each segment or pseudo-node simulating the edge. Figure 3.11 shows a saturated edge along with a selected node disk. The following

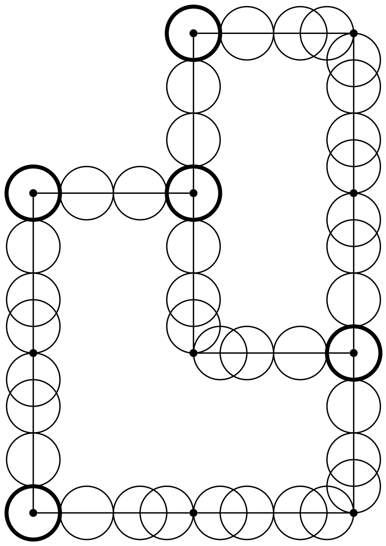


Figure 3.10: The embedding has 5 nodes, 3 bending pseudo-nodes, and 3 non-bending pseudo-nodes. For easier identification, the figure outlines in bold the disks that simulate nodes, and shows the underlying grid graph embedding.

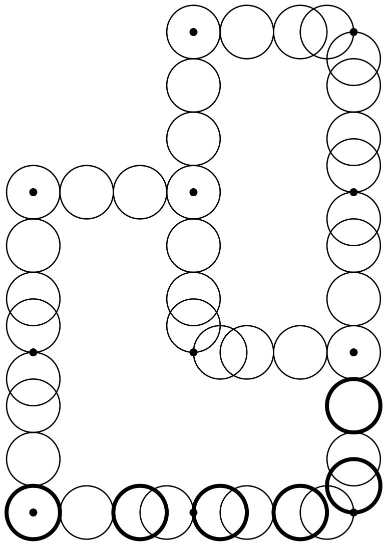


Figure 3.11: A saturated edge and a selected node disk

observation follows from the construction of the simulation.

Observation 3.24 *An edge in the planar cubic graph G_{pc} can be saturated if and only if at most one of its node disks is selected.*

Now suppose that I_{pc} is an independent set of k vertices in G_{pc}. Select the node disks corresponding to I_{pc}. By the observation above, we can now saturate every edge of G_{pc} by selecting exactly one disk from each segment and pseudo-node in such a way that no two disks overlap. This process yields k' non-overlapping disks, which make up I_{ud} as required.

Conversely, suppose that $G_{ud}(P)$ has an independent set of k' disks, and let I_{ud} be the maximum independent set that maximizes the number of saturated edges. Then every selected node disk v is adjacent only to saturated edges. For otherwise the number of saturated edges could be increased: remove v and the less than $e/2$ disks in the adjacent unsaturated edge from I_{ug}, and replace them with the $e/2$ disks that saturate the edge, as shown in Figure 3.12. By the observation, I_{pc}, the set of vertices in G_{pc} corresponding to node disks in I_{ud}, is an independent set. Since the edges can contribute at most $p+s$ disks to the k' disks in I_{ud}, it follows that $k' \leq |I_{pc}|+p+s$. That is, $|I_{pc}| \geq k'-p-s = k$, as required.

Squares rotated by 45 degrees can substitute for disks in the above proof. If the grid graph is first rotated by 45 degrees, then upright squares can substitute for the disks, also. Therefore the proof is valid for disks under the L_∞ and L_1 metrics, as well as the L_2 metric. ▮

Corollary 3.24.1 *VERTEX COVER is* **NP***-complete for unit disk graphs under the L_1, L_2, and L_∞ metrics.*

Proof: VERTEX COVER has the same complexity as INDEPENDENT SET problem with respect to restrictions on the graph [GJ79]. ▮

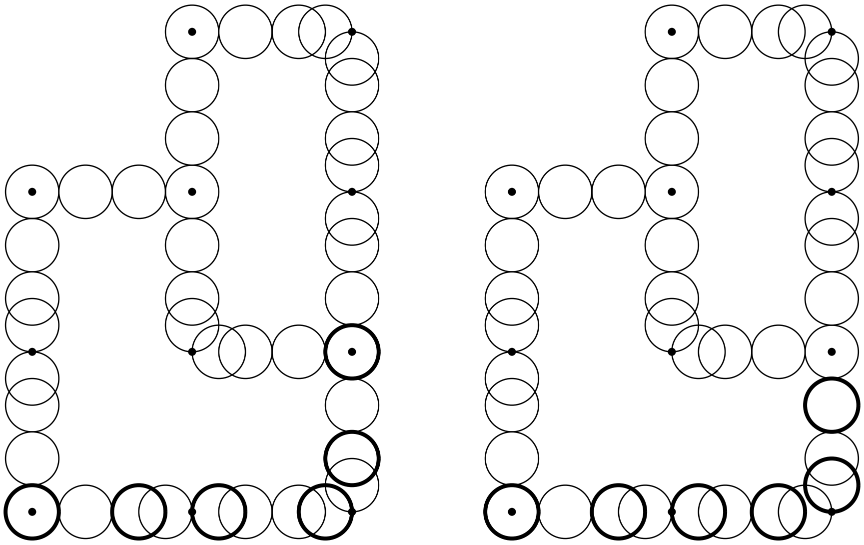


Figure 3.12: Saturating an unsaturated edge. The bold circles on the left form a maximal independent set of disks from the edge. However, the edge can be saturated by redistributing one of the node disks into the edge, as shown on the right.

3.3.2 Maximum Clique

Finding a maximum clique in an arbitrary graph is a well-known **NP**-complete problem [GJ79]. This problem is also closely related to finding a maximum independent set, since a clique in a graph is an independent set in its complement. We have already seen that INDEPENDENT SET is **NP**-complete for unit disk graphs (Section 3.3.1). Nevertheless, we will see in this section that a maximum clique in a unit disk graph can be found in polynomial time, given a realization of the graph.

Let $G = (V, E)$ be a unit disk graph and let $f : V \to \mathbf{R}^2$ be a realization of G. To simplify the notation, let us assume that vertex names are synonymous with their realizations; that is, $v = f(v)$ for every vertex $v \in V$. Although this practice might lead to ambiguity, the role of a vertex will always be clear from context. First note that if C is a maximum clique with more than one vertex in G, then it has a pair of sites p and q that are farther apart than any other pair in C. The distance Δ between p and q must naturally satisfy $\Delta \leq 1$. Furthermore, all sites in C must be within Δ units of p and q. That is, let *the lune through points* p *and* q be the set of points in the region $R_{p,q}$, where

$$R_{p,q} = \{x : x \in \mathbf{R}^2 \text{ and } \|x - p\| \leq \Delta \text{ and } \|x - q\| \leq \Delta\}$$

as shown in Figure 3.13. Then $C \subset R_{p,q}$, and in particular, $C \subseteq L_{p,q}$, where $L_{p,q} = V \cap R_{p,q}$ is the *discrete lune* through p and q.

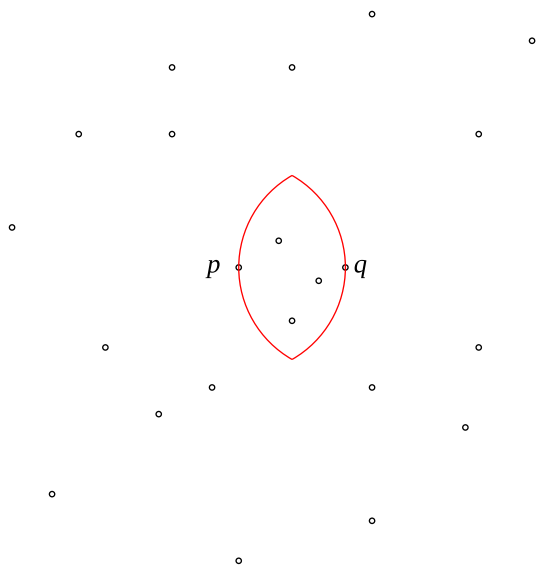


Figure 3.13: The lune through a pair of sites.

A maximum clique containing p and q is therefore a maximum clique in the induced subgraph $G(L_{p,q})$. We can now compute a maximum clique in $G(L_{p,q})$ by computing a maximum independent set in the graph's complement $\overline{G}(L_{p,q})$, since these sets are identical.

Clark, Colbourn, and Johnson [CCJ90] prove that $G(L_{p,q})$ is cobipartite, that is, that $\overline{G}(L_{p,q})$ is bipartite. This is easy to see. The line through p and q partitions the lune through p and q into two halves, say an upper half and a lower half. It is easy to verify that each half has diameter Δ, which is at most 1. Therefore, any edge in the complement must join a vertex in one half with a vertex in the other. Clark *et al.* show how to extract a maximum independent set from a maximum matching in $\overline{G}$. Furthermore, a maximum matching in a bipartite graph can be found in $O(V^{2.5})$ time ([HK73], see for example [PS82] pages 225–226).

One algorithm for finding a maximum clique in a unit disk graph $G(V) = (V, E)$, therefore, is to examine all $|E|$ adjacent pairs of vertices in V. For each pair $\{p, q\}$, compute $\overline{G}(L_{p,q})$ in $O(V^2)$ time. Finally, compute a maximum independent set of the subgraph $\overline{G}(L_{p,q})$ in $O(V^{2.5})$ time. The complete algorithm therefore runs in $O(V^{2.5}E) = O(V^{4.5})$ time. This proves the following theorem.

Theorem 3.25 ([CCJ90]) *One can find a maximum clique of a unit disk graph $G = (V, E)$ in $O(V^{4.5})$ time, given a realization.*

We can actually do better if we exploit the realization. The following algorithm is similar to one developed by Aggarwal *et al.* [AIKS89] for finding k points, from n points in the plane, with minimum diameter. Aggarwal *et al.* proved independently of [CCJ90] that $\overline{G}(L_{p,q})$ is bipartite. They then use an algorithm due to Imai and Asano [IA86], together with an efficient representation of $\overline{G}(L_{p,q})$ due to Hershberger and Suri [HS89], to find a maximum independent set in $\overline{G}(L_{p,q})$ in $O(V^{1.5} \log V)$ time. The algorithm in the previous paragraph, together with the observations in this paragraph, therefore takes a total time of $O(V^{3.5} \log V)$.

Another result due to Imai and Asano [IA83] is an algorithm that finds a maximum clique of an intersection graph of rectangles in $O(V \log V)$ time.

Since a unit disk graph under L_1 and L_∞ is an intersection graph of squares, their algorithm applies here, also. The following theorem summarizes the preceding discussion.

Theorem 3.26 *Given a realization, one can find a maximum clique in a unit disk graph $G = (V, E)$ in $O(V^{3.5} \log V)$ time under the L_2 metric and in $O(V \log V)$ time under the L_1 and L_∞ metrics.*

Is it possible to find a maximum clique in a unit disk graph *without* a realization? For example, Gräf ([Grä95] pages 35–36) interprets [CCJ90] to conclude that $L_{p,q} = N(p) \cap N(q)$, thereby obviating the need for a realization. Unfortunately, this conclusion is unwarranted if $\Delta < 1$. The intersection of neighbourhoods $N(p) \cap N(q)$ is indeed enclosed by a (geometric) lune, but this lune is the intersection of two unit-radius disks. As shown in Figure 3.14, this unit-radius lune is larger than the lune through p and q, and the graph induced on its sites may not be cobipartite.

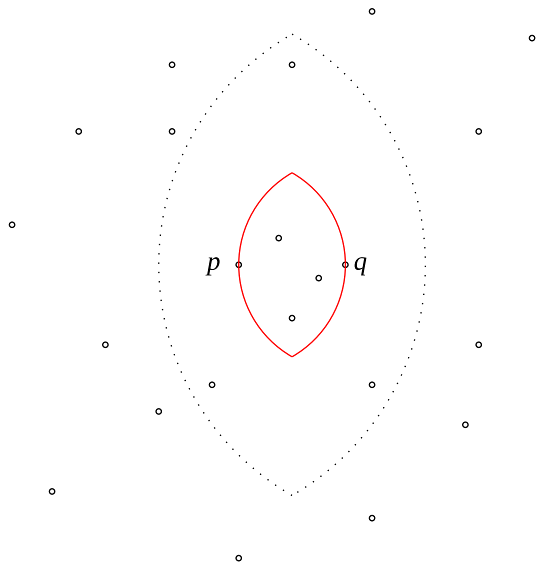


Figure 3.14: The unit-lune defined by a pair of sites, together with the smaller lune through the pair of sites.

Perhaps if p and q have the greatest separation of any pair in some clique, then the subgraph induced by $N(p) \cap N(q)$, although not a cobipartite *lune*, is nevertheless still cobipartite (for other reasons). Then the maximum clique algorithm could be easily modified to work without a realization. This conjecture also is not true, as shown by the more detailed counterexample in Figure 3.15. The question "Can the maximum clique of a unit disk graph be computed in polynomial time without access to a realization?" therefore remains open.

3.3.3 Chromatic Number

To *colour* a graph, assign a positive integer (a *colour*) to each vertex in the graph, such that no two adjacent vertices receive the same colour. The *minimum colour* problem is to colour a graph using the least number of colours. This least number of colours is called the *chromatic number* of the graph. The related decision problem, GRAPH K-COLOURABILITY, is **NP**-complete for arbitrary graphs [GJ79]. We will see in this section that it remains **NP**-complete when restricted to unit disk graphs.

Section §2.1.4 presented an illustrative application to radio frequency spectrum management due to Hale [Hal80]. Hale states that the colouring problem for unit disk graphs was shown to be **NP**-complete by J.B. Orlin in April of 1980, but does not provide any further details or references. Since then it has been proved by Burr [Bur82] and by Clark, Colbourn, and Johnson [CCJ90]. An independent proof due to the author appears below. It follows the general framework outlined earlier, of reducing from the same problem in planar graphs with degree at most four.

GRAPH K-COLOURABILITY (Problem GT4 in [GJ79])

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.

QUESTION: Is G K-colourable, that is, does there exist a function

$$c : V \to \{1, 2, \ldots, K\} \text{ such that } c(u) \neq c(v) \text{ whenever } (u, v) \in E?$$

Theorem 3.27 (Orlin 1980 [Hal80, Bur82, CCJ90]) *GRAPH K-COLOURABILITY is* **NP***-complete for unit disk graphs, even for* $K = 3$.

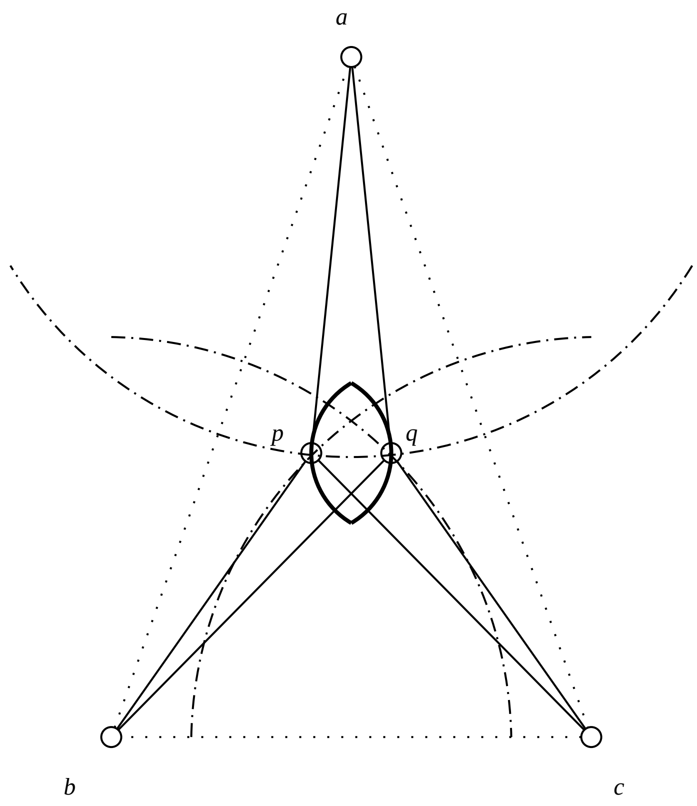


Figure 3.15: $G(N(p) \cap N(q))$ is not always cobipartite. The figure shows unit-radius arcs centered at a, b, and c. For clarity, the figure does not show an edge between p and q, even though these vertices are adjacent. Site p is exactly unit distance from sites a and c. Similarly, q is exactly unit distance from a and b. Therefore, $\{a, b, c\} \subset N(p) \cap N(q)$. The remaining sites (not shown in the figure) fall into one of two halves. The *upper half* lies in the lune, just above the unit-radius arcs from both b and c. The *lower half* also lies in the lune, just below the unit-radius arc from a. These additional sites are placed such that p and q have the greatest separation. By this construction, all points in the lune form a clique. In fact, they form a maximum clique; vertex a cannot be part of a maximum clique, since its presence would exclude the entire lower half. Similarly, neither b nor c can be part of a maximum clique, since the presence of either one would exclude the entire upper half. Finally, note that $G(N(p) \cap N(q))$ is not cobipartite since $\{a, b, c\}$ induces a triangle in its complement.

Proof: The problem is in **NP** since it is in **NP** for arbitrary graphs. **NP**-hardness follows by a reduction from GRAPH 3-COLOURABILITY restricted to planar graphs having no vertex degree exceeding 4 [GJ79]. Let $H = (V, E)$ be an instance of GRAPH 3-COLOURABILITY. Figure 3.16 shows an example graph H. Begin, as usual, by em-

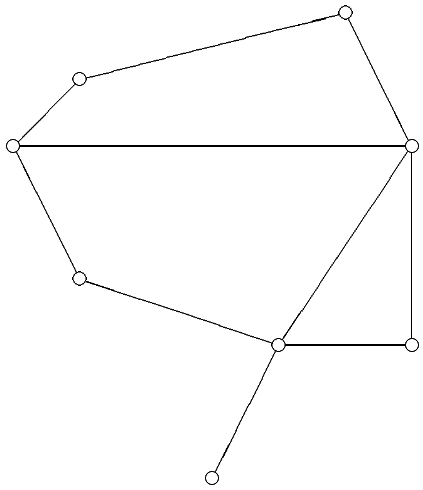


Figure 3.16: A planar graph with degree at most 4.

bedding H into a square grid in quadratic time and space using Valiant's method (§3.3.1). This embedding assigns integer coordinates to each vertex in H. Such an embedding is shown in Figure 3.17, where the vertices of H are drawn as concentric circles. The embedding introduces new vertices, which we will call *pseudo-vertices*, and which Figure 3.17 shows as single circles. Valiant's embedding typically dilates an edge in H (i.e., in E) into several unit length edges. For each edge in E, arbitrarily select one of the resulting edges and call it a "real" edge. Figure 3.17 shows these as dotted line segments. Call the remaining unit-length edges, "pseudo-edges"; Figure 3.17 shows these as solid line segments.

Next, multiply all coordinates by 12, in effect refining the grid. We are now ready to construct an instance of a unit disk graph $G_4(P)$ generated by a set of points P. Let P

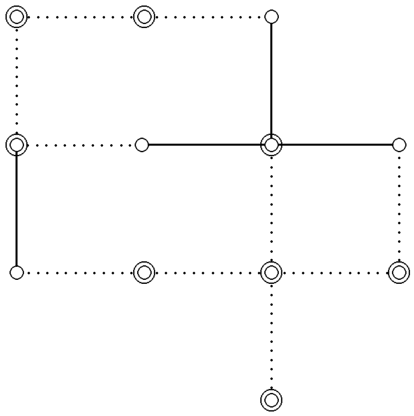


Figure 3.17: The graph from Figure 3.16 embedded in the grid.

include points with the coordinates of all vertices, both real and pseudo. In addition, P includes points corresponding to the edges. Simulate edges with the points (subgraphs) in Figure 3.18. Note that any 3-colouring of pseudo-edges results in vertices with the

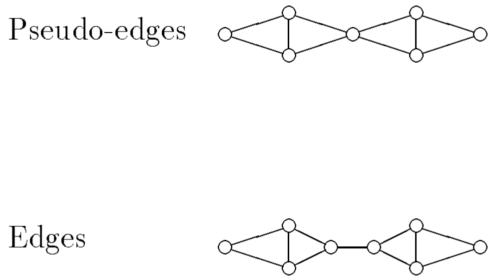


Figure 3.18: Simulations for edges and pseudo-edges. Note that, in any 3-colouring, the endpoints of real edges must get different colours, and the endpoints of pseudo-edges must get the same colour.

same colour, and any 3-colouring of real edges results in vertices with different colours. Pseudo-edges may therefore be conveniently thought of as extensions of the incident real vertices. Furthermore, the vertices adjacent to real edges can be coloured by any pair of non-identical colours (in the absence of other constraints). Figure 3.19 shows

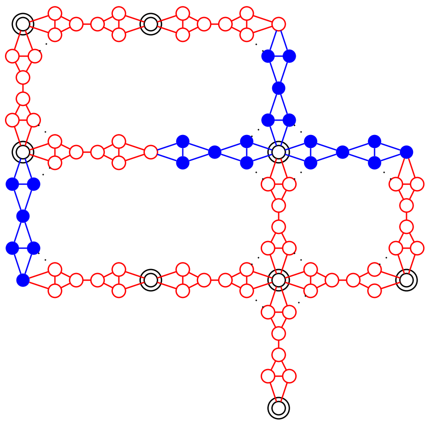


Figure 3.19: An instance of unit disk graph 3-colourability. This figure shows "real" vertices as concentric circles. It shows "real" edges as open circles, and "pseudo-edges" as solid disks.

a complete transformation of the graph in Figure 3.16. Note that this construction introduces additional graph edges, which Figure 3.19 depicts as dotted line segments. These are artifacts of the "drawing". In fact, the vertical pairs of points in Figure 3.18 can be arbitrarily close together (but then they would would not be easily distinguishable in the drawing), thereby avoiding the connections in Figure 3.19. Nevertheless, it is easy to see that if the graph with these edges removed can be 3-coloured, then so can the entire graph.

If $c : V \to \{1, 2, 3\}$ is a 3-colouring of G, then c restricted to the real vertices is a 3-colouring of H, since the two kinds of edge component enforce different colours for adjacent real vertices. Similarly, if $c : V \to \{1, 2, 3\}$ is a 3-colouring of H, then c can be extended to the vertices of G. Hence G is 3-colourable if and only if H is 3-colourable. ∎

Approximation Algorithms

Even though GRAPH K-COLOURABILITY remains **NP**-complete for unit disk graphs (Theorem 3.27), we can use the greedy colouring algorithm to colour a unit disk graph with no more than three times the minimum number of colours. If we are required to colour each vertex exactly when it is presented to us "on-line", then the greedy algorithm still manages to use at most five times the minimum. Note that, for arbitrary graphs G, Lund and Yannakakis [LY93] have shown that there is some constant $\epsilon > 0$ such that no polynomial-time graph colouring algorithm has a performance ratio $R \leq |V(G)|^{\epsilon}$ (unless $\mathbf{P} = \mathbf{NP}$). It particular, for arbitrary graphs, unlike unit disk graphs, R cannot be a constant. Much of this subsection appears in a joint paper with Marathe, Hunt, Ravi, and Rosenkrantz [MBH+95]. In particular, most unattributed lemmas appear in [MBH+95] also.

The well-known *greedy colouring algorithm* simply colours each vertex with the first available colour. The algorithm colours each vertex in some order, so assume that the set of vertices V is totally ordered. For clarity of analysis and exposition, assume that each edge of the graph is directed from earlier to later vertices in the order of V. Table 3.8 presents the algorithm in detail.

Table 3.8: **Algorithm:** GREEDY(G) [Colour G sequentially with the first available colour]

Input: A totally ordered, directed unit disk graph $G = (V, A)$
Output: The number of colours used to colour G.
Side Effects: colour[v] is the colour of each vertex.

for $v \leftarrow$ first vertex in V **to** last vertex in V
  do colour[v] $\leftarrow \min(\mathbf{Z}^{+} \setminus \{\text{colour}[u] : (u, v) \in A\})$
return $\max \{\text{colour}[v] : v \in V\}$

Lemma 3.28 *Algorithm GREEDY colours (arbitrary) graphs with no more than one plus the maximum indegree colours. That is, GREEDY*$(G) \leq \Delta_{in}(G) + 1$.

Proof: The algorithm maintains the invariant that any assigned colour is at most $\Delta_{in}+1$. The invariant holds trivially at the beginning. Thereafter, vertex v has indegree at most Δ_{in} and outdegree 0, so vertex v too can be coloured with one of the $\Delta_{in}+1$ colours not used by its neighbours. ∎

Corollary 3.28.1 *Algorithm GREEDY colours (arbitrary) graphs with no more than one plus the maximum degree colours. That is, GREEDY$(G) \leq \Delta(G)+1$.*

An On-Line Algorithm

In order to analyze the behaviour of Algorithm GREEDY on unit disk graphs, we need to relate its degree to its chromatic number. One way to do this is by relating the degree of a unit disk graph to its maximum clique size, and relating this to its chromatic number. This results in Theorem 3.30 below.

Lemma 3.29 *The maximum clique size of a unit disk graph G is greater than one sixth of its maximum degree. That is, $\omega(G) \geq \lfloor \Delta(G)/6 \rfloor + 2$.*

Proof: Let v be a vertex with the greatest degree. Its neighbours lie in the unit radius disk centered on the vertex. Therefore more than one sixth, at least $\lfloor \Delta/6 \rfloor + 1$ of these neighbours, lie in some 60 degree unit sector. Since such a sector has unit diameter, these neighbours together with the vertex v form a completely connected subgraph. A maximum clique must be at least as large. ∎

Theorem 3.30 *Algorithm GREEDY colours unit disk graphs with less than six times the maximum clique size. That is, GREEDY$(G) \leq 6\omega(G)-6$.*

Proof: Lemma 3.29 implies that $\omega \geq \lfloor \Delta/6 \rfloor + 2 > \Delta/6+1$ so that $\Delta < 6\omega - 6$. Since both Δ and ω are integers, $\Delta \leq 6\omega - 7$. By Corollary 3.28.1, we have

$$\text{GREEDY}(G) \leq \Delta + 1 \leq 6\omega - 7 + 1 = 6\omega - 6.$$

▮

Corollary 3.30.1 *Algorithm GREEDY colours unit disk graphs with less than six times the optimal number of colours. That is, GREEDY(G)* $\leq 6\chi(G) - 6$.

Proof: This follows from Theorem 3.30 since $\chi \geq \omega$ for all graphs. ▮

However, this corollary is overly pessimistic. Using the following lemma from [MBH+95], Gräf [Grä95] obtains a tighter bound on the chromatic number, thereby showing (Corollary 3.32.1) that Algorithm GREEDY performs even better than stated by Corollary 3.30.1.

Lemma 3.31 *Let G be a unit disk graph. Any independent set in the neighbourhood of a vertex in G has at most five vertices.*

Proof: If the lemma were false, then some vertex would be the hub of an induced star $K_{1,6}$, thereby contradicting Lemma 3.1, which says that there can be no such star in a unit disk graph. ▮

Lemma 3.32 ([Grä95]) *The chromatic number of a unit disk graph G is greater than one fifth of its maximum degree. That is,* $\chi(G) \geq \lceil \Delta(G)/5 \rceil + 1$.

Proof: Let v be any vertex in a unit disk graph $G = (V, E)$. By Lemma 3.31, at most five vertices adjacent to v can get the same colour. Therefore $\chi(G(\mathrm{Adj}(v)) \geq \lceil |\mathrm{Adj}(v)|/5 \rceil$. Since v is adjacent to every vertex in its neighbourhood, it must be coloured differently than its neighbours. That is, $\chi(G(\mathrm{N}(v))) = \chi(G(\mathrm{Adj}(v))) + 1$ (recall that $\mathrm{N}(v) = \mathrm{Adj}(v) \cup \{v\}$). Consequently,

$$\chi(G) \geq \chi(G(\mathrm{N}(v))) = \chi(G(\mathrm{Adj}(v))) + 1 \geq \lceil |\mathrm{Adj}(v)|/5 \rceil + 1.$$

The lemma follows if we choose v to have the maximum degree. ▮

Corollary 3.32.1 ([Grä95]) *Algorithm GREEDY colours a unit disk graph with less than five times its chromatic number. That is, GREEDY(G)* $\leq 5\chi(G) - 4$.

Proof: Lemma 3.32 states that $\chi(G) \geq \lceil \Delta(G)/5 \rceil + 1 \geq \Delta(G)/5 + 1$, so that $\Delta(G) \leq 5\chi(G) - 5$. By Corollary 3.28.1, we have $\text{GREEDY}(G) \leq \Delta + 1 \leq 5\chi - 5 + 1 = 5\chi - 4$. ∎

An Off-Line Algorithm

In an "off-line" setting, we have the luxury of examining the entire graph before colouring any vertices. We will continue to exploit Algorithm GREEDY, but we will do so by reordering the vertices to our advantage. Essentially, we will create an order for the vertices so that no vertex has indegree greater than three times the graph's chromatic number. Say that (the vertex set of) a unit disk graph G is *lexicographically ordered* if $x_f(u) < x_f(v)$ (or $x_f(u) = x_f(v)$ and $y_f(u) < y_f(v)$) implies $u < v$, for some realization $f : V \to \mathbf{R}^2$. Gräf attributes the first use of this order for colouring unit disk graphs to Peeters [Pee91].

Lemma 3.33 ([Pee91, MBH+95, Grä95]) *The maximum clique size of a lexicographically ordered unit disk graph is greater than one third of the maximum indegree. That is,* $\omega(G) \geq \lceil \Delta_{in}(G)/3 \rceil + 1$.

Proof: Let v be a vertex with the greatest indegree. Its (indegree) neighbours lie in the left unit-radius semicircle centered on the vertex. Therefore at least one third, $\lceil \Delta_{in}/3 \rceil$, of these neighbours lie in a 60 degree unit sector. Since such a sector has unit diameter, these neighbours, together with the vertex v, form a completely connected subgraph. The maximum clique must be at least as large. ∎

Theorem 3.34 ([Pee91, MBH+95, Grä95]) *Algorithm GREEDY colours lexicographically ordered unit disk graphs with less than three times the maximum clique size. That is,*

$$\mathit{GREEDY}(G) \leq 3\omega(G) - 2.$$

Proof: Lemma 3.33 implies that $\omega \geq \lceil \Delta/3 \rceil + 1 \geq \Delta/3 + 1$ so that $\Delta \geq 3\omega - 3$. By Lemma 3.28, we have GREEDY$(G) \leq \Delta + 1 \leq 3\omega - 3 + 1 = 3\omega - 2$. ∎

Corollary 3.34.1 ([MBH+95]) *Algorithm GREEDY colours lexicographically ordered triangle-free unit disk graphs with at most four colours.*

Proof: The maximum clique size of a triangle-free graph is at most 2. ∎

Corollary 3.34.2 ([Pee91, MBH+95, Grä95]) *Algorithm GREEDY colours lexicographically ordered unit disk graphs with less than three times the optimal number of colours. That is, GREEDY$(G) \leq 3\chi(G) - 2$.*

Proof: This follows from Theorem 3.34 since $\chi \geq \omega$ for all graphs. ∎

Note that we required a realization to lexicographically order the vertices. We can dispense with the realization as follows. Gräf points out that Algorithm GREEDY will perform at least as well given *any* vertex ordering that achieves a maximum indegree no greater than that of the lexicographic order. In particular, he mentions that Matula [MB83] has shown that the smallest-last ordering minimizes the maximum indegree over all vertex orderings. The vertices $\{v_1, v_2, \ldots, v_n\}$ of a graph are said to be in *smallest-last order* if v_i has minimum degree in $G(\{v_1, v_2, \ldots, v_i\})$ for all i. There is an easy algorithm for finding a smallest-last ordering: find a minimum degree vertex v in the graph $G = (V, E)$, recursively smallest-last order $G(V \setminus \{v\})$, and append v to the end of the sequence. Matula [MB83] shows how to implement this algorithm in $O(V + E)$ time with $O(V)$ additional space.

Gräf's PhD thesis [Grä95] also presents a more sophisticated algorithm (for colouring unit disk graphs) that still uses a realization, and still has a theoretical performance ratio of 3, but which performs well in practice. See also [MBH+95] for a different algorithm that achieves the same performance ratio of 3, but does not require a realization.

In closing this section on colouring, recall that all triangle-free disk graphs are planar (Theorem 3.4). Therefore such graphs can be coloured with four colours by the famous four-colour theorem [AH76]. This improves on Lemma 5.2 in [MBH+95], which states that six colours suffice for triangle-free disk graphs.

3.4 Unit Disk Graph Recognition is NP-Hard

This section proves that recognizing unit disk graphs is **NP**-hard. Equivalently, it shows that determining if a graph has sphericity 2 or less, even if the graph is planar or is known to have sphericity at most 3, is **NP**-hard. In fact, this section gives a polynomial-time reduction from SATISFIABILITY to a more general problem, that of recognizing ρ-bounded disk graphs, which involve disks of restricted sizes. It begins by describing the reduction for ρ-bounded coin graphs, where the disks have pairwise-disjoint interiors. We will see that this reduction can be extended to three dimensions, thereby showing that unit sphere graph recognition, or determining if a graph has sphericity 3 or less, is also **NP**-hard.

Section 3.4.1 defines SATISFIABILITY and ρ-bounded disk and coin graphs. Thereafter, Sections 3.4.2 through 3.4.9 show how to reduce SATISFIABILITY to the problem of recognizing ρ-bounded coin graphs. Finally, Section 3.4.10 shows how to extend the result to other problems, including the already-mentioned problem of recognizing unit disk graphs.

3.4.1 Coin Graphs

A *coin* is a closed disk in the plane. A graph G is called a *coin graph* if G is the intersection graph of a set of interior-disjoint coins [Sac94]. Remarkably, finite coin graphs are precisely the finite planar graphs [Sac94], and can therefore be recognized in linear time (see [BL76] for example). In fact, every plane graph G has a coin realization $f(G)$ such that $(v_1, v_2, \ldots, v_k)$ is a clockwise face in G if and only if $(f(v_1), f(v_2), \ldots, f(v_k))$ is a clockwise face in $f(G)$.

In this section, however, we are interested in the complexity of recognizing graphs than can be realized with coins of bounded size. A set of disks is *ρ-bounded* if every disk in the set has diameter between 1 and ρ inclusive. We will prove that ρ-BOUNDED COIN GRAPH RECOGNITION is **NP**-hard. A special case of this problem is recognizing *penny graphs*, where all disks have unit diameter. Penny graphs where all unit disks must be centered at integer coordinates are equivalently grid graphs (§2.3.3), which are also a subclass of unit disk graphs. Recall from Section 2.3.3 that GRID GRAPH RECOGNITION is **NP**-complete [BC87], even for grid graphs that are binary trees [Gre89].

We will have occasion to deal with the location and the radius of disks separately. To ease notation, say that $f : V \to \mathbf{R}^2$ (locations) and $r : V \to \mathbf{R}$ (disk radii) constitute a *disk realization* (f, r) of a graph $G = (V, E)$ if $(u, v) \in E$ if and only if $\|f(u) - f(v)\| \leq r(u)+r(v)$. Then a disk realization (f, r) is a *coin realization* if $\|f(u)-f(v)\| \geq r(u)+r(v)$ for all u and v in V, adjacent or not, and it is *ρ-bounded* if $1 \leq 2r(v) \leq \rho$ for all $v \in V$. This section shows that the following parameterized recognition problem is **NP**-hard for any fixed $\rho \geq 1$.

ρ-BOUNDED COIN GRAPH RECOGNITION

INSTANCE: A graph $G = (V, E)$.

QUESTION: Is G the intersection graph of a set of closed, interior-disjoint disks whose diameters fall in the range $[1, \rho]$, that is, does G have a ρ-bounded coin realization?

Since coin graphs are planar, they do not include K_5. Since any intersection graph class must contain all complete graphs, this means that coin graphs are *not* the intersection graph class of any family of sets [Sch85], in particular not of any family of disks. On the other hand, this section's reduction of SATISFIABILITY to ρ-BOUNDED COIN GRAPH RECOGNITION extends easily to allow overlapping disks as well. That is, recognizing ρ-bounded disk intersection graphs (equivalently the intersection graph class of ρ-bounded disks) is also **NP**-hard for any fixed $\rho \geq 1$. The special case $\rho = 1$ of this latter problem (namely UNIT DISK GRAPH RECOGNITION) has been reported separately [BK93].

ρ-BOUNDED DISK GRAPH RECOGNITION

INSTANCE: A graph $G = (V, E)$.

QUESTION: Is G the intersection graph of a set of closed disks whose diameters fall in the range $[1, \rho]$, that is, does G have a ρ-bounded disk realization?

The problem CNF SATISFIABILITY defined below is a common basis for **NP**-completeness proofs [Coo71, GJ79]. Let $U = \{u_1, u_2, \ldots, u_m\}$ be a set of Boolean *variables.* A clause $c = \{l_1, l_2, \ldots, l_k\}$ is a set of literals, which are negated (e.g., $\bar{u}_i$) and unnegated (e.g., u_i) variables from U. A set C of clauses is intended to represent the conjunctive normal form Boolean formula

$$\mathcal{F} = \bigwedge_{c \in C} \bigvee_{l_i \in c} l_i.$$

A *truth assignment* is a function $t : U \to \{\text{TRUE}, \text{FALSE}\}$. In terms of literals, u_i is TRUE if and only if $t(u_i) = \text{TRUE}$, and $\bar{u}_i$ is TRUE if and only if $t(u_i) = \text{FALSE}$. A clause is *satisfied* by t if at least one $l_i \in c$ is TRUE. Finally, a *satisfying truth assignment for C* is one that simultaneously satisfies all the clauses in a set C.

[LO1, [GJ79]] **CNF SATISFIABILITY**

INSTANCE: A set $U = \{u_1, u_2, \ldots, u_m\}$ of Boolean variables and
a set $C = \{c_1, c_2, \ldots, c_n\}$ of clauses over U.

QUESTION: Is there a satisfying truth assignment for C?

Theorem 3.35 *ρ-BOUNDED COIN GRAPH RECOGNITION is* **NP***-hard for any fixed $\rho \geq 1$.*

Proof: Given an instance C of SATISFIABILITY, we will construct a graph $G_C = (V_C, E_C)$ such that G_C has a realization[3] if and only if C is satisfiable. Assume without loss of generality (see comments for SATISFIABILITY in [GJ79]) that each clause in C contains *at most* three literals ($|c_i| \leq 3$) and that each variable appears in *at most* three clauses. Note that this is not the same as 3SAT in which every clause has exactly three literals, and each variable appears in an unrestricted number of clauses (though 3SAT remains **NP**-complete if every variable appears in at most five clauses [GJ79]).

We will build G_C in several stages. First, in Section 3.4.2, we will construct a bipartite graph G_C^{SAT} that corresponds closely to the instance C of SATISFIABILITY. We will define a notion of orientability for this graph, and prove that it is orientable if and only if C is satisfiable (Lemma 3.37). Then, in Section 3.4.3, we will draw this graph on the square grid. This natural drawing on the grid maintains the intuitive structure of the SATISFIABILITY problem. It also prevents edge drawings from "interfering" with

[3]For brevity, and unless stated otherwise, the remainder of this section abbreviates "ρ-bounded coin graph realization" as "realization".

each other by keeping them rectilinear and crossing them only at right angles. We will define a notion of orientability for this drawing. This serves to express the graph as a composition of smaller graphs, each of which is a primitive for orientation. The drawing is orientable if and only if the underlying graph is orientable (Lemma 3.39). Next, we will skew the grid so that it is part of the triangular grid. This will allow us to build and connect components by exploiting a hexagonal packing. Finally, we will form G_C in Section 3.4.4 by simulating components of the skewed grid drawing. Lemma 3.42 shows that G_C has a realization if and only if the underlying grid drawing is orientable. We finish by showing that the entire reduction can be executed in polynomial time. ∎

3.4.2 A Graph That Simulates SATISFIABILITY

Our first step is to construct a bipartite graph G_C^{SAT} from the instance C of SATISFIABILITY as follows. The vertices of the graph correspond to the clauses, variables, and negated variables of the SATISFIABILITY instance C. There is an edge between a literal vertex and a clause vertex if the literal appears in the clause. More formally, $G_C^{SAT} = (V_C^{SAT}, E_C^{SAT})$, where:

$$\begin{aligned} V_C^{SAT} &= \{\tilde{c} : c \in C\} \cup \{u^+ : u \in U\} \cup \{u^- : u \in U\} \\ E_C^{SAT} &= \{(\tilde{c}, u^+) : c \in C, u \in c\} \cup \{(\tilde{c}, u^-) : c \in C, \bar{u} \in c\}. \end{aligned}$$

Definition 3.36 The graph G_C^{SAT} is *orientable* if its edges can be directed such that, (1) for each clause vertex, outdegree($\tilde{c}$) $\geq$ 1 and, (2) for each pair of literal vertices, indegree(u^+) $= 0$ or indegree(u^-) $= 0$. ∎

The graph G_C^{SAT} models the testing of a truth assignment for SATISFIABILITY by directing its edges. Intuitively, an edge directed from $\tilde{c}$ to u^+ (resp. u^-) means that clause c has selected literal u (resp. $\overline{u}$) to satisfy it. That is, clause c requests that $t(u) =$ TRUE (resp. $t(u) =$ FALSE).

Lemma 3.37 *The set of clauses C is satisfiable if and only if the bipartite graph G_C^{SAT} is orientable.*

Proof: If t satisfies C, then orient G_C^{SAT} by directing every edge incident on u^+ towards u^+ if $t(u) =$ TRUE, or away from u^+ otherwise. Similarly, direct every edge incident on u^- towards u^- if $t(u) =$ FALSE, or away from u^- otherwise. Then outdegree$(\tilde{c}) \geq 1$ for every clause c, since c must be satisfied. Furthermore, either indegree$(u^+) = 0$ or indegree$(u^-) = 0$ for every variable u, since no truth assignment can set both a literal and its complement TRUE. Therefore G_C^{SAT} is orientable if C is satisfiable.

Conversely, if G_C^{SAT} has been oriented, then set $t(u) =$ TRUE if any edge $(\tilde{c}, u^+)$ is directed towards u^+. Similarly set $t(u) =$ FALSE if any edge $(\tilde{c}, u^-)$ is directed towards u^-. Then t sets each variable either TRUE or FALSE since indegree$(u^+) = 0$ or indegree$(u^-) = 0$ for all variables u. Furthermore, t must satisfy every clause c since outdegree$(\tilde{c}) \geq 1$ for every vertex $\tilde{c}$. ∎

3.4.3 Drawing the Graph on the Grid

Our next step is to draw the graph G_C^{SAT} on the grid as shown in Figure 3.20. Each of the $(6|U| + 1) \times (3|C| + 2)$ grid vertices in this drawing is either unused, or is associated with a unique *component* of the drawing. Each component is enclosed by a unit square centered on its own grid vertex.

The drawing is made up of three groups of components: communication components, literals, and clauses. There are, in turn, three groups of communication components: wires, corners, and crossovers. A *wire* is a unit length line segment passing through a grid vertex, a *corner* is two half-length line segments meeting at right angles at a grid vertex, and a *crossover* is two unit-length line segments crossing at right angles on a grid vertex. There are therefore two types of wire components (horizontal and vertical), four types of corners, and one type of crossover.

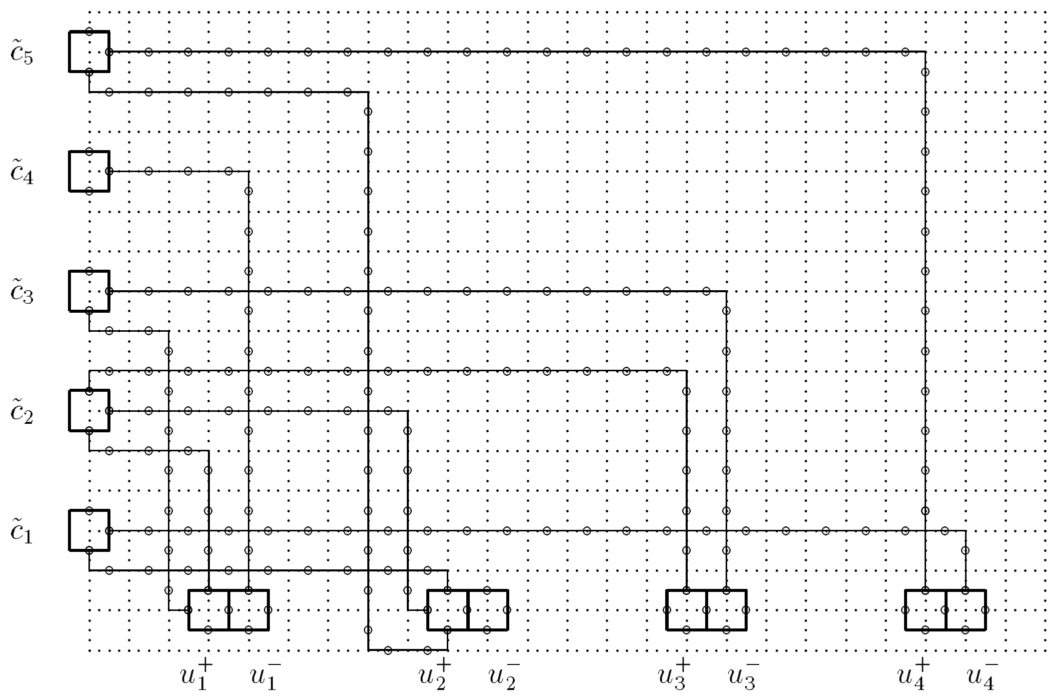


Figure 3.20: A SATISFIABILITY graph drawn on the grid. The graph corresponds to the SATISFIABILITY instance (U, C), where $U = \{u_1, u_2, u_3, u_4\}$, $C = \{c_1, c_2, c_3, c_4, c_5\}$, and $c_1 = \{u_2, \bar{u}_4\}$, $c_2 = \{u_1, u_2, u_3\}$, $c_3 = \{u_1, \bar{u}_3\}$, $c_4 = \{\bar{u}_1\}$, and $c_5 = \{u_2, u_4\}$. The clauses and the literals are drawn as squares. The variables are embedded as adjacent pairs and so are drawn as rectangles. Each clause and each literal component has three terminals. Note that the area of the grid is $(6|U| + 1) \times (3|C| + 2)$.

Each component in the drawing has two to four *terminals*, which are centered on a side of the unit square enclosing the component. The terminal on the top (or north) side of the unit square is called the T terminal. Similarly, the terminals on the bottom, left, and right are called the B, L, and R terminals respectively. Wires and corners have two terminals each, crossovers and literals have four, and clauses have three. Two components in the drawing are *adjacent* if they have coincident terminals. The terminal between two adjacent literals is called the (common) *interliteral* terminal; the others are called *external* terminals. Figure 3.20 depicts the set of all terminals as small circles.

An *orientation* of a terminal is a *direction*, North, South, East, or West. Say that a terminal T (respectively B, L, R) is *directed away* from its component if it is oriented North (respectively South, West, East) and is *directed towards* its component otherwise.

Definition 3.38 A grid drawing is *orientable* if all terminals can be oriented subject to the following four conditions.

draw1: T and B terminals must be directed North or South,

draw2: L and R terminals must be directed East or West,

draw3: every wire, corner, crossover line segment, and clause must have at least one terminal directed away from it,

draw4: If a literal has its interliteral terminal directed towards it, then *all* other (external) terminals must be directed away from it.

∎

A corollary of conditions **draw4** and **draw2** is that if a drawing is orientable, then every variable has a literal component with all external terminals directed away from it. Figure 3.21 shows an orientation of a portion of Figure 3.20.

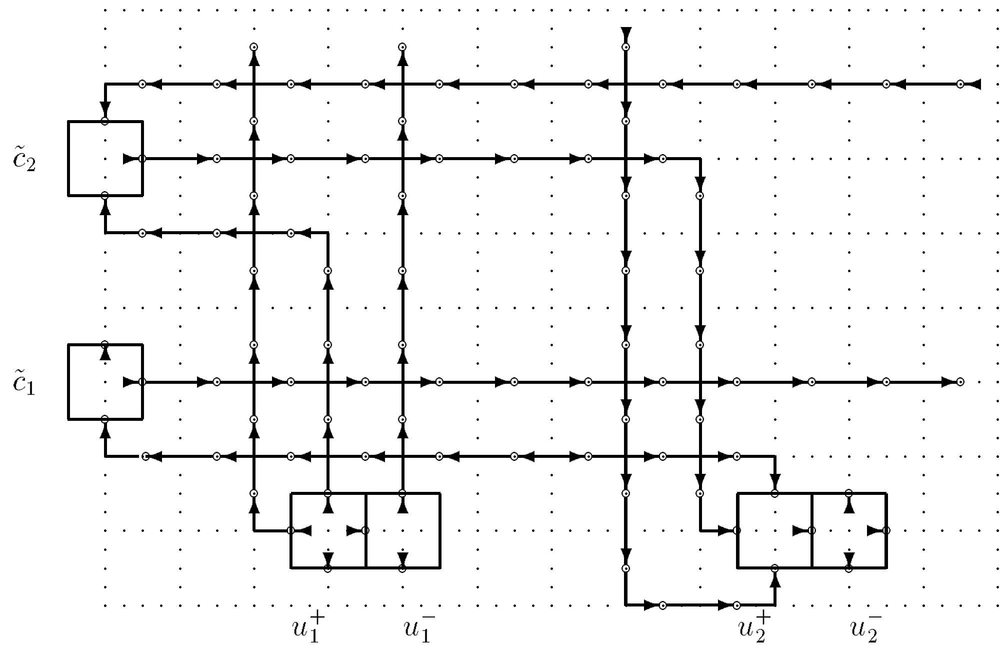


Figure 3.21: An oriented grid drawing. The directions are drawn as arrows. Note that the path from u_2^+ to $\tilde{c}_1$ has a wire component that has both terminals directed away from it.

Lemma 3.39 *A grid drawing is orientable if and only if the underlying bipartite graph is orientable.*

Proof: Suppose we have an orientation for G_C^{SAT}. Then orient the terminals along each path[4] in the grid drawing so that they are consistent with the orientation of the corresponding edge in E_C^{SAT}. This ensures that every wire, corner, and crossover line segment has *precisely* one terminal directed away from it. It also ensures that each clause has one terminal directed away from it, since the clause vertices have outdegree at least one. Finally, every variable has a literal component with all external terminals directed away from it since one of the literal vertices has zero indegree in G_C^{SAT}; direct the interliteral terminal towards this literal.

Now suppose that the grid drawing has been oriented. Condition **draw3** ensures that

[4]Those terminals not on a path need not concern us; they may be arbitrarily oriented, say away from the component.

any path in the drawing corresponding to an edge in E_C^{SAT} contains at most one wire, corner, or crossover line segment with both terminals directed away from the component. One terminal of such a component can be redirected by reversing the direction of all terminals in the path from it to the literal. Doing so keeps all conditions satisfied since this operation does not change the orientation of any clause terminals, and it directs any external literal terminals away from the literal, in keeping with condition **draw4**. Condition **draw3** then ensures that all terminals along the path are oriented consistently with some orientation for the corresponding edge. Under this orientation, all clause vertices have outdegree at least one, since the corresponding terminal is directed away from the clause. Furthermore, condition **draw4** ensures that indegree(u^+) $= 0$ or indegree(u^-) $= 0$. ∎

Corollary 3.39.1 *A grid drawing is orientable if and only if the underlying instance of SATISFIABILITY is satisfiable.*

Skewing the Grid

It would be easier to simulate the grid drawing components with coin subgraphs if G_C^{SAT} had been embedded in a triangular grid instead of a square grid. This is because coins (disks) of the same size naturally pack hexagonally. We will exploit this packing for the two extreme sizes for the disks, that is, for diameter 1 and ρ. Hexagons also have a higher connectivity (six) with their neighbours than do squares (four), allowing us to construct more compact components. Both of these advantages will be evident in the forthcoming constructions. We chose to embed G_C^{SAT} on a square grid to capture the intuition behind the bipartite SATISFIABILITY graph. Fortunately, it is easy to turn the square grid into a triangular one by "skewing" it as follows. If we imagine that the square grid is generated by two basis vectors $(1, 0)$ and $(0, 1)$, then we can skew the grid by replacing

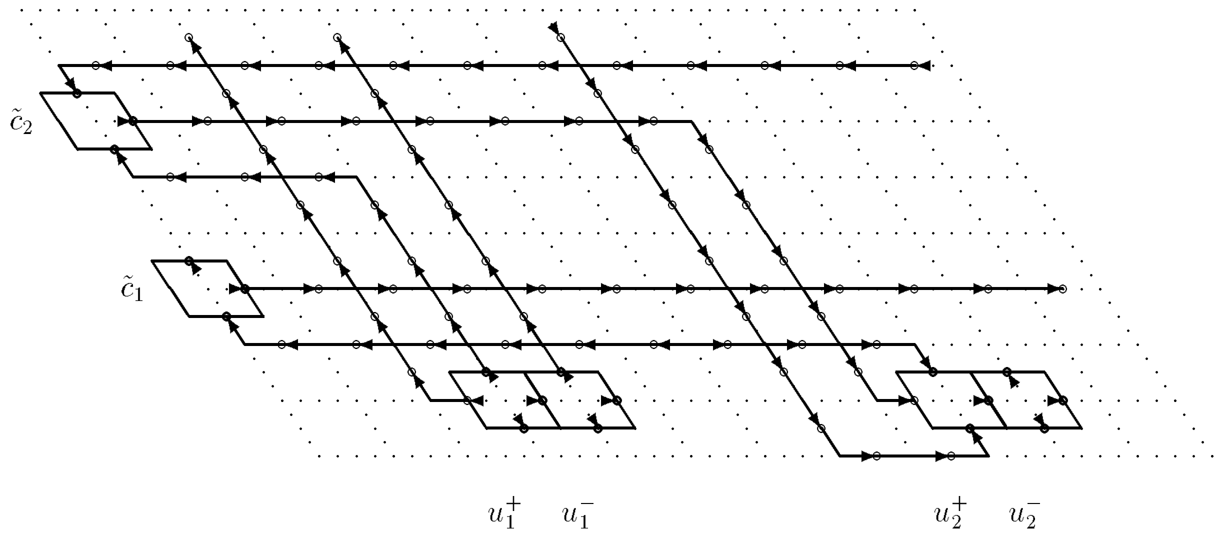


Figure 3.22: A skewed version of the oriented grid drawing from Figure 3.21.

these with the new basis vectors $(1, 0)$ and $(-1/2, \sqrt{3}/2)$. The result of skewing the drawing from Figure 3.21 is shown in Figure 3.22. To aid visibility, the figure does not show grid lines with positive diagonal slope, but imagine that they are there to complete the triangular grid.

3.4.4 Simulating the Drawing with Coins

We are now ready to construct G_C. To do so, we create a graph component for each skewed grid drawing component: wire, corner, crossover, literal, and clause. Section 3.4.8 provides the details. To specify the graph, we will describe a realization of each of its components, and then describe how the subgraphs are to be connected. Therefore, before giving detailed descriptions of the various graph components, we must examine the building blocks from which they are constructed. The main building blocks are cycles, called "cages". The construction joins two cages at a shared adjacent triple of vertices to create larger components. The remaining building blocks are clusters of vertices called "flippers". Flippers are associated with the vertex triple shared by two cages. In any realization, all the vertices in a flipper are forced to lie in one of the two incident cages. To ensure this consistent embedding, the vertices of a flipper are connected among themselves. Flippers come in three different sizes measured as a fraction of a cage's capacity: 1, 1/2, and 1/4. These sizes, and the capacity of a cage, allow us to rule out certain embeddings.

Figure 3.23 shows schematically how cages and flippers are related. The next sub-

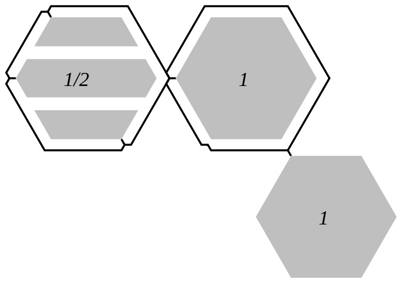


Figure 3.23: Schematic drawings for cages and flippers. This schematic shows two joined cages. The one on the left contains two quarter flippers and a half flipper. The one on the right contains a full flipper, and has another attached full flipper that has been displaced. The labels on the flippers indicate their asymptotic portion of the hexagonal capacity. Unlabelled flippers are quarter flippers. Note the dimpled connecting corners, as well as one dimpled nonconnecting corner.

section provides detailed instruction for constructing cages and flippers, and proves the required properties. The schematic shows a *realization* of a subgraph, but we are really constructing the *subgraphs* themselves. Therefore, let the schematics depict also the underlying coin graph. The realizations therefore serve both to specify the graph, and to show how the graph could be realized.

A flipper embedded inside a cage diminishes the capacity of the cage for additional flippers. It thereby displaces other flippers, which may otherwise have been embedded inside the cage. The forthcoming construction (§3.4.7) for G_C ensures that a flipper can only be embedded in one of two adjacent cages. A displaced flipper therefore displaces other flippers from whatever cage it is embedded in. This is the basic mechanism for propagating information in a realization of G_C.

Constructing Cages

Cages will always have the same number of vertices, which depends to some extent on the diameter ρ of the largest disk. However, this number will be a multiple of six, so that a cage can be realized as a string of disks whose centres lie on a regular hexagon aligned with the triangular grid, as shown in Figure 3.24. Say that such a realization is

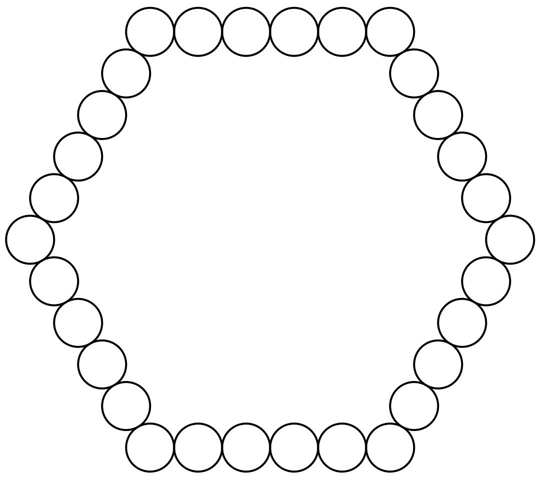


Figure 3.24: The hexagonal realization of a cage.

hexagonal. In the construction that follows, we join cages at the corners of such hexagonal realizations, as shown in Figure 3.25. Clearly, both cages cannot be strictly hexagonal in such a construction. To solve this problem, we choose to *dimple a corner* by moving the corner disk towards the centre of one of the hexagons. Naturally, the corner may be dimpled towards either of its two hexagons. Clearly, we can dimple a corner even if it is not shared with another cage; we will attach flippers to corners in this way.

Define the *hexagonal capacity* of a cage as the maximum number of interior-disjoint disks that can be packed hexagonally into a hexagonal realization of a cage, as shown in Figure 3.26. Note that the capacity depends only on the number of cage vertices.

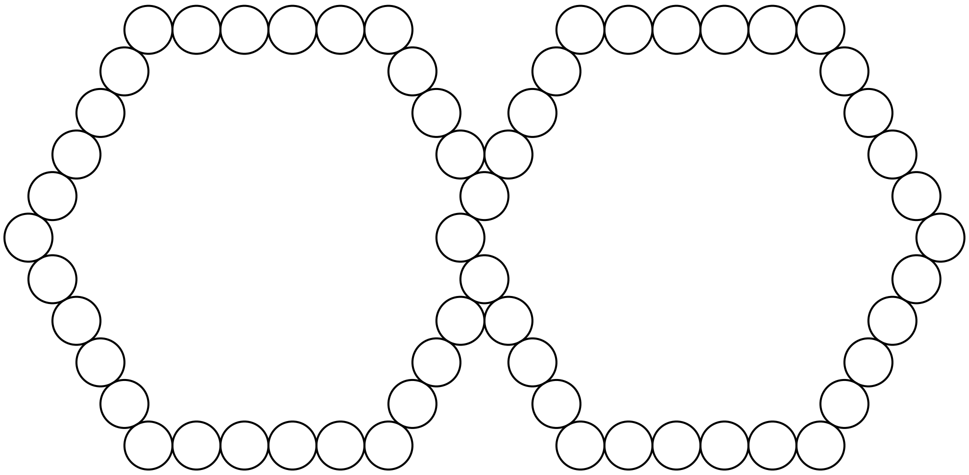


Figure 3.25: Joining cages. The corner of the cage on the left is "dimpled". The alternative hexagon realization, in which the corner is dimpled to the right, is also permitted.

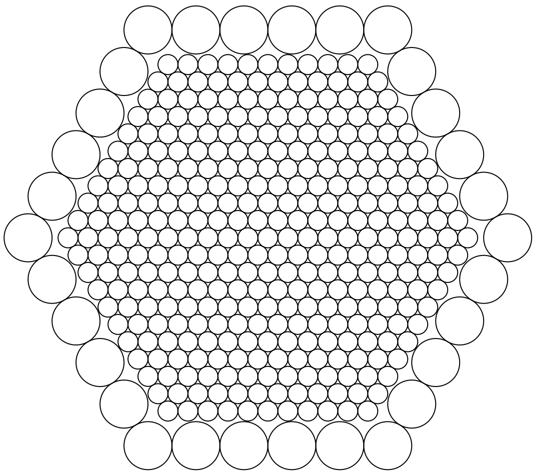


Figure 3.26: A hexagonal realization of a cage packed hexagonally with unit-diameter disks. The cage disks all have maximum diameter ρ.

3.4.5 Fabricating Flippers

We are now ready to fabricate flippers for cages. We will do so "asymptotically". Perhaps this is somewhat unusual, but the idea is simple: fabricate a flipper for an arbitrary cage size, and then notice that the size of the flipper approaches an ideal value as the cage size increases. Therefore, although our flipper never actually attains this value, we will see that it attains a suitable value.

Full Flippers

In any realization, a flipper must intersect (touch) the cage only at one corner, even if some of the other corners are dimpled. To ensure that this is the case, dimple in *all* corners of the cage. Fabricate a *full flipper* as follows. Pack the dimpled cage with the largest hexagon of unit disks in contact with the desired corner. Make sure that the flipper is not in contact with any other parts of the cage, including the dimpled corners, as shown in Figure 3.27.

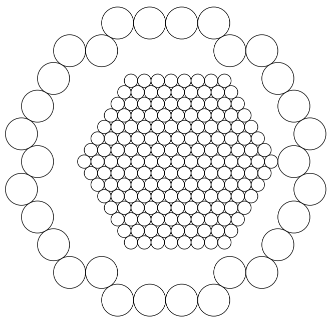


Figure 3.27: Fabricating full flippers

The number of disks in a full flipper approaches the hexagonal capacity of the cage as the size of the cage increases. This is because the size of any disk becomes negligible in comparison with the sides of the cage. In particular, the gap between the flipper and the cage becomes negligible. Note that, even though the required cage size may be large, it does not depend on the size of the SATISFIABILITY instance.

Half Flippers

There are two kinds of half flippers. The fabrication of quarter flippers, described in the next paragraph, results in one kind. To fabricate the other kind, notice that the line segment through the hexagon in Figure 3.28.(a) divides the area of the hexagon into two equal areas. Construct a *half flipper* from this drawing and a full flipper by removing any

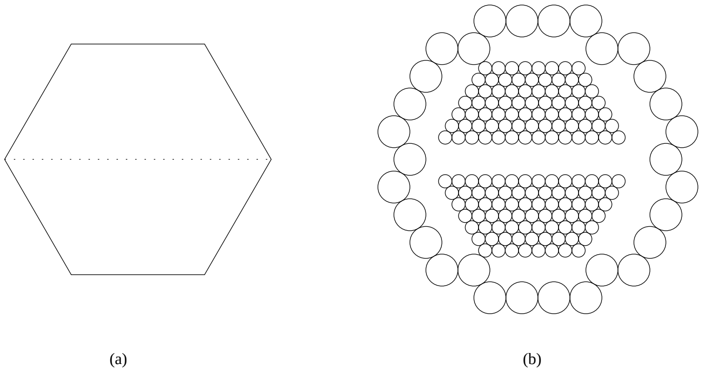


Figure 3.28: Fabricating a half flipper

disks hit by the line segment, as well as the half hexagon not in contact with the cage. Figure 3.28.(b) shows two half flippers constructed in this way. As the size of the cage (and the size of the packing) increases, the number of disks in the half flipper approaches one half of the hexagonal capacity.

Quarter Flippers

Similarly, the line segment through the hexagon in Figure 3.29.(a) divides the area of the hexagon into two regions. The smaller region has one quarter of the hexagon's area, and the larger region has three quarters the area. Fabricate a flipper that approaches 1/4 of the hexagonal capacity from this drawing by removing from a full flipper any disks hit by the line segment, as well as the three quarters hexagon not in contact with

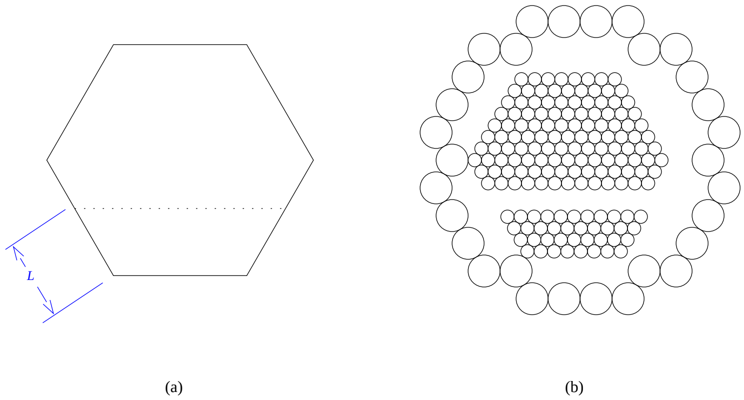


Figure 3.29: Dividing a hexagon into a quarter and three-quarters. The length L of the short side of the small trapezoid is $(\sqrt{10}-2)/4$, which ensures that the area of the trapezoid is one-quarter that of the hexagon with unit sides.

the cage. Similarly, fabricate a flipper that approaches 3/4 of the hexagonal capacity by removing any disks hit by the line segment, as well as the one quarter hexagon not in contact with the cage. Figure 3.29.(b) shows both such flippers. As the cage size increases, the number of disks in the small section approaches one quarter the hexagonal capacity, and the number of disks in the other section approaches three quarters of the hexagonal capacity.

In the same way, fabricate another kind of half flipper by adding a second segment to the hexagon, symmetric with the first as shown in Figure 3.30.(a). Figure 3.30.(b) shows the result.

3.4.6 Capacity Arguments

The flipper fabrications concentrated solely on the hexagonal capacity of a cage. In fact, these capacities are close to optimal. Define the *unconstrained capacity* of a cage as the maximum number of interior-disjoint disks that can be packed into any realization of a cage. Again, the capacity depends only on the number of cage vertices. We may as well assume that all disks in a maximum packing have unit diameter, since replacing any

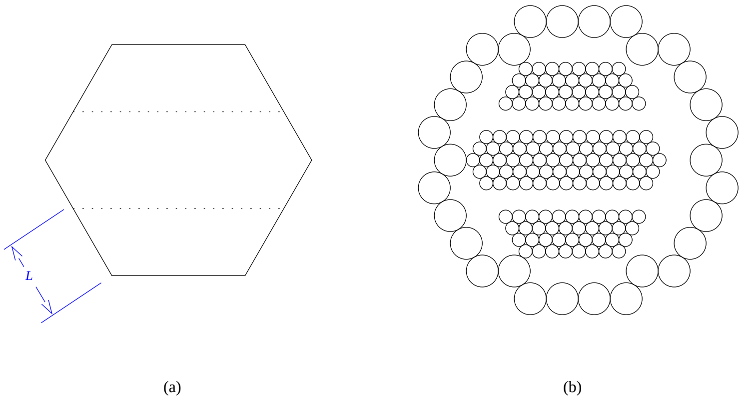


Figure 3.30: Dividing a hexagon into two quarters and a half

larger disks with unit disks results in an interior-disjoint packing with the same number of disks (and possibly room for more). With the help of the following lemma, we will be able to prove that some embeddings cannot be realized.

Lemma 3.40 (The Capacity Lemma) *The unconstrained capacity of all sufficiently large cages is less than* $12/\pi^2$ *(*≈ 1.216*) times its hexagonal capacity.*

Proof: We will construct an asymptotic value for the hexagonal capacity C_h of a cage, and a (rather conservative) upper bound for its unconstrained capacity C_u. We will then see that $C_u < (12/\pi^2)C_h$.

First, consider a hexagonal realization of a cage, and pack it also hexagonally, see Figure 3.26 again. As the size of the cage increases, its perimeter[5] P approaches that of its corresponding hexagonal packing. That is, $P = 6s$ in the limit, where s is the length of a hexagon side. Note that s is one less than the number of unit disks on one of the hexagon's sides. It is easy to verify by induction on s that the total number of disks in the packing, which is the hexagonal capacity C_h of the cage, is given by Equation 3.6.

$$C_h = 3s^2 + 3s + 1 > 3s^2 \tag{3.6}$$

[5]The *perimeter* of a string of disks is just the perimeter of the polygon through its disk centres. The perimeter of a set of disks is the perimeter of the convex hull of its centres.

To construct an upper bound on the unconstrained capacity C_u, notice that the interior of a cage can never be completely covered with interior-disjoint unit disks, since unit disks do not tile the plane. Therefore C_u is less than the maximum internal area, taken over all possible realizations of the cage, divided by the area of the unit disks. As the size of a cage increases, its internal area approaches from below the area of the polygon through its disk centres. The circle has maximum area over all polygons with the same perimeter. Therefore, the realization of a cage as a set of maximum-size disks with cocircular centres achieves, at the limit, the maximum internal area. As noted, this area is less than the area of the circle through its disk centres. Calculate this circle's area as follows. First note that the circumference c of the circle is the same as the perimeter $P = 6s$ of the hexagonal realization. Therefore the radius r of the circle is

$$r = P/2\pi = 3s/\pi,$$

so that its area A is

$$A = \pi r^2 = \pi(3s/\pi)^2 = 9s^2/\pi.$$

Since the area of a unit-diameter disk is $\pi/4$, it follows that

$$\begin{aligned} C_u &< \frac{A}{\pi/4} \\ &= \frac{9s^2/\pi}{\pi/4} \\ &= \frac{36s^2}{\pi^2}. \end{aligned} \tag{3.7}$$

Combining Inequalities 3.7 and 3.6 yields the following inequality, which proves the lemma.

$$\frac{C_u}{C_h} < \frac{36s^2/\pi^2}{3s^2} = \frac{12}{\pi^2}$$

∎

Corollary 3.40.1 *In any realization, no sufficiently large cage can contain a subset of flippers whose total nominal size exceeds* $12/\pi^2$.

Note that this corollary does not exploit the fact that a hexagonal disk packing is asymptotically optimal [CS88]. By virtue of this corollary, and our earlier requirement for hexagonal realizations, the main building block for the ensuing construction is a cage whose size (number of vertices) is a sufficiently large multiple of six. To emphasize the asymptotic nature of these cages, the rest of this **NP**-hardness reduction abbreviates cages and flippers by the schematics shown previously in Figure 3.23.

3.4.7 The Skeleton of the Graph G_C

The following lemma, which applies to arbitrary disk intersection graphs as well as coin graphs, shows that the *inside* of a realization of a cycle is well defined.

Lemma 3.41 *A realization of a vertex-induced cycle is a plane graph.*

Proof: If the realization were not a plane graph, then two line segments would cross somewhere. Then, by Lemma 3.3, the endpoints of the segment would induce a triangle in the graph. This contradicts the fact that cages are triangle-free. ∎

The skeleton of the graph under construction will be composed of many cages "hooked together", as presented already in Figure 3.25. Lemma 3.3 and its corollary guarantees that cages do not cross or overlap each other in any realization. In addition, the attached flippers and the capacity of the cages keeps cages from containing each other. This connection strategy therefore ensures that, in any realization, the clockwise order of the edges about every cage is determined by the order of the edges about *any* cage in the connected graph. It is therefore useful to think of the embedding of the skeleton of the graph, that is, the cages without flipper vertices, to be invariant under all realizations.

Different realizations simply allow flippers to "flip" from one cage into another. Flippers must be in one of their two incident cages, by virtue of their construction and Lemma 3.3. The component definitions which follow will clarify this construction.

The properties described above allow realizations of unit disk graphs to simulate the orientability of the grid drawing, depending on which cage encloses which flippers. In addition, we must ensure that the skeleton of the graph, that is, the graph induced on the cages, is always realizable. In particular, the components must "fit together". For example, they must stretch between two grid vertices (from Figure 3.20). To achieve this requirement, each component joins together several cages. For example, the forthcoming wire component consists of five cages. The number five comes from the realization scheme described in Section 3.4.9.

3.4.8 The Components of the Graph G_C

Each of the graph components has two or more *terminals*—labelled T, B, L, or R—that correspond to the grid drawing terminals. Each terminal is a three-vertex corner of a cage with an attached full flipper. For example, the three-vertex corners belonging to terminals L and R are circled in Figure 3.31 below. To construct G_C, connect every pair of adjacent[6] components together by identifying the appropriate terminals. Terminal T (resp. B, L, R) should be identified with an adjacent B (resp. T, R, L) terminal. You will find that the example presented in Section 3.4.9 clarifies this process.

Wires

The graph in Figure 3.31 implements the horizontal wire component as a string of five cages joined by corners (refer back to Figure 3.25).

[6]Two graph components are *adjacent* if the associated grid drawing components are adjacent.

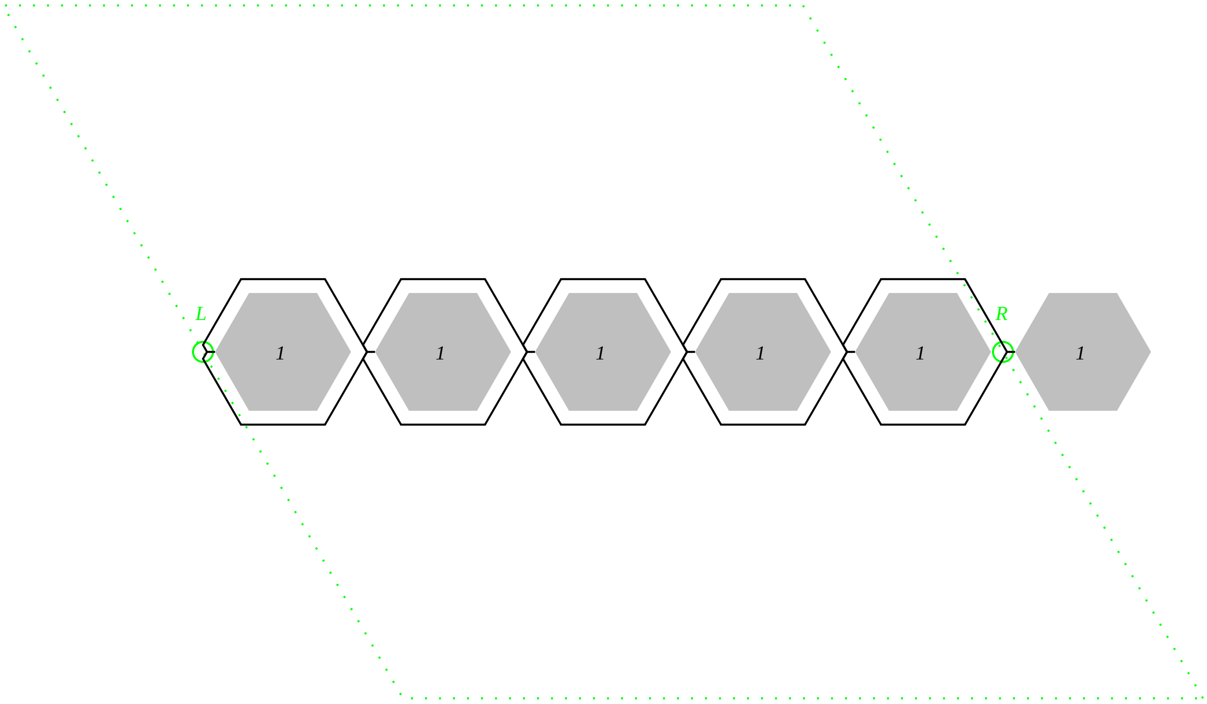


Figure 3.31: The horizontal wire component: drawn with both terminals oriented East. The terminals are indicated by small circles. Remember that the terminals include the flippers. The grid square enclosing the wire component is drawn as a dotted parallelogram.

Say that the T (resp. B, L, R) terminal is *oriented* South (resp. North, East, West) if its flipper is embedded inside its cage, and that it is oriented North (resp. South, West, East) otherwise. The orientation of the terminals shown in Figure 3.31, or in any of the following figures, is not the only possible orientation. However, the properties of the cages ensure that, for any realization, if the L terminal is oriented East, then its R terminal is also oriented East. Similarly, if the R terminal is oriented West, then its L terminal is also oriented West. This ensures that at least one terminal is directed away from the wire. Note that it is also possible for the L and R terminals to be simultaneously oriented West and East respectively.

The vertical wire component is shown in Figure 3.32. Note that although the realizations are different, and the terminals are labelled differently, the (unlabelled) horizontal and vertical wire component subgraphs are isomorphic. Again, for any realization, if either terminal is oriented inwards, then the other one must be oriented outwards.

Corners

As can be seen from Figure 3.33, there are four kinds of corners, and their graphs are also strings of five cages. Again, the properties of cages ensure that at least one terminal is directed away from each corner in any realization.

Crossovers

Of all components used in this reduction, the crossover component is the most difficult to understand. Its schematic is shown in Figure 3.34. We need to convince ourselves that paired terminals (that is, T and B, or L and R) cannot be simultaneously oriented towards the component, but that all other orientations are possible.

The heart of the crossover is the ring of cages labelled (a, b, c, d, e, f) in Figure 3.34. The *flippers associated with a set of cages* are those that are shared by two cages of

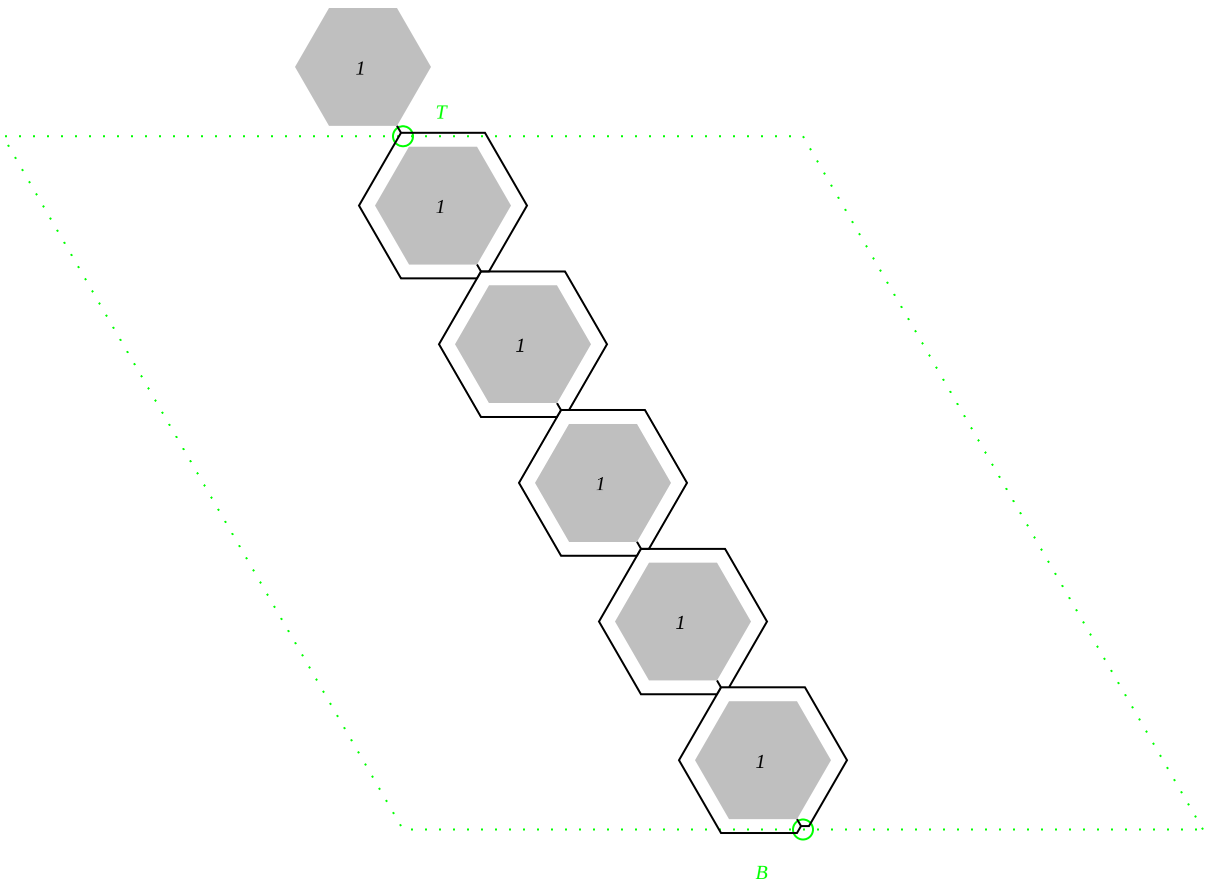


Figure 3.32: The vertical wire component: drawn with both terminals oriented North.

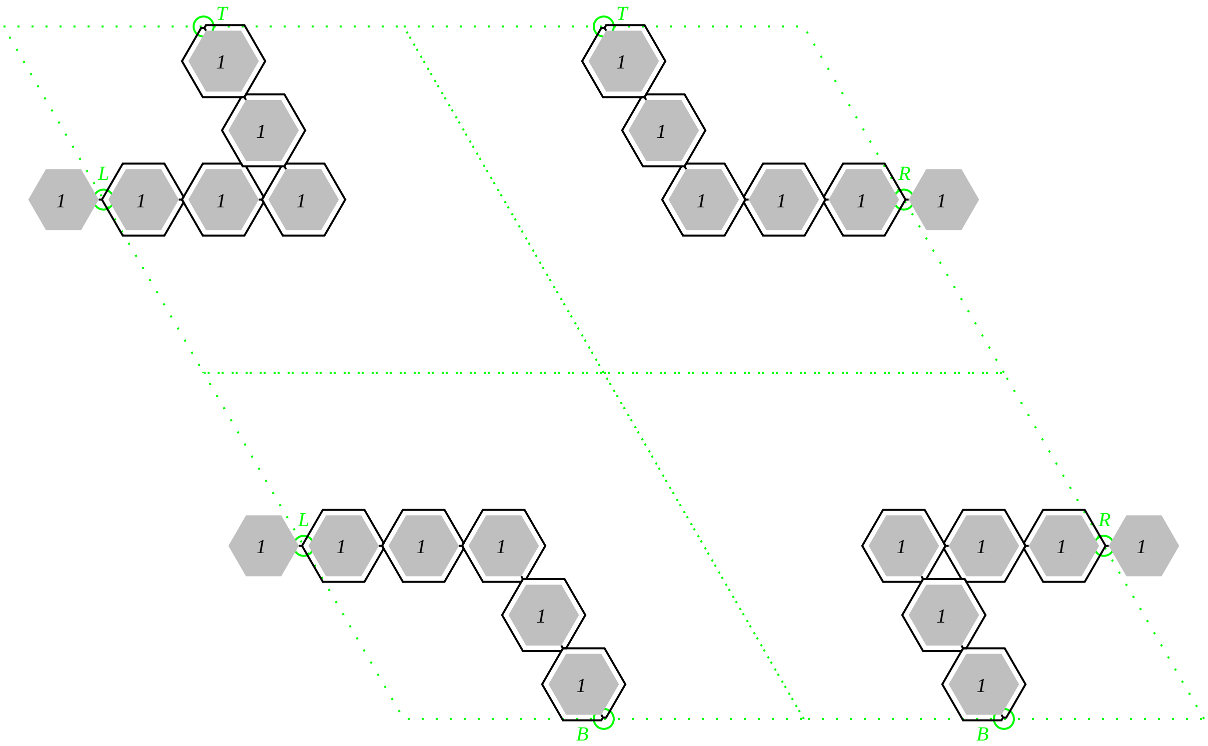


Figure 3.33: The corner components. They are drawn with the L terminals oriented West, the R terminals oriented East, the T terminals oriented South, and the B terminals oriented North. Note that not all four are isomorphic, since on two of them, some cages are connected but do not share flippers.

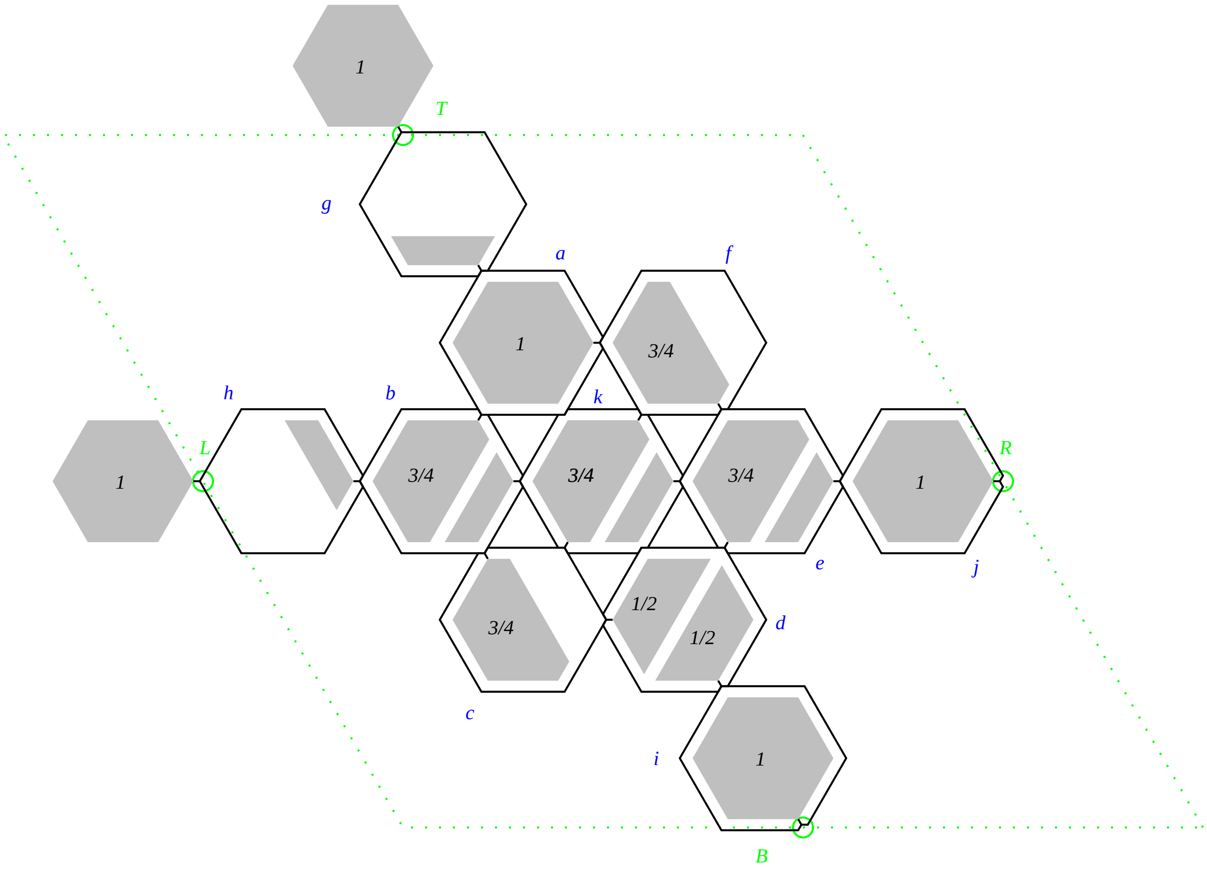


Figure 3.34: The crossover component. The T and B terminals are drawn oriented North, and the L and R terminals are drawn oriented West. The ring of cages (a, b, c, d, e, f) is drawn oriented counterclockwise.

the set. In any realization, the flippers associated with the ring can take on one of two orientations: counterclockwise, as shown in Figure 3.34, or clockwise. This is because the number (6) of cages and associated flippers is the same. By Corollary 3.40.1, no two flippers can be in the same cage. Consequently, every cage of the ring must contain exactly one of its shared flippers. Note that this means that the three-quarters flipper in the central cage k must remain in this cage for all realizations. Figure 3.34 emphasizes this fact by showing two connections for this flipper, but of course one will suffice.

The crossover allows two channels to cross without interference. The horizontal channel transmits its information via the horizontal chain of cages, and the diagonal channel transmits its information via the ring. Let us examine the horizontal channel in action. Since cages b, k, and e must each contain a three-quarters flipper, none of them can contain two additional quarter flippers, by Corollary 3.40.1. This means that the quarter flippers associated with the horizontal cages (h, b, k, e, j) can take on one of two orientations, west or east. The west orientation, shown in Figure 3.34, displaces the full flipper from cage h, and the east orientation displaces the full flipper from cage j. Conversely, if terminal R is oriented West, then it forces the west orientation on (h, b, k, e, j), and if terminal L is oriented East, then it forces the east orientation on (h, b, k, e, j).

Let us now examine the diagonal channel in action. If the ring (a, b, c, d, e, f) is oriented counterclockwise as shown, then the full flipper in cage a displaces the quarter flipper into cage g. If it is oriented clockwise, then the three-quarters flipper displaces the half flipper from cage d. Conversely, if terminal B is oriented North as shown, then it forces the counterclockwise orientation on the ring, and if terminal T is oriented South, then it forces the clockwise orientation on the ring. Finally, the orientation of the ring, and the orientation of the horizontal channel, are independent. To see this, create the four possible orientations of the crossover by "swivelling" the flippers on their articulation points in Figure 3.34. Figure 3.39 later in this section shows all four orientations in use.

Literals

The positive literal component is shown in Figure 3.35. Its heart is the central cage,

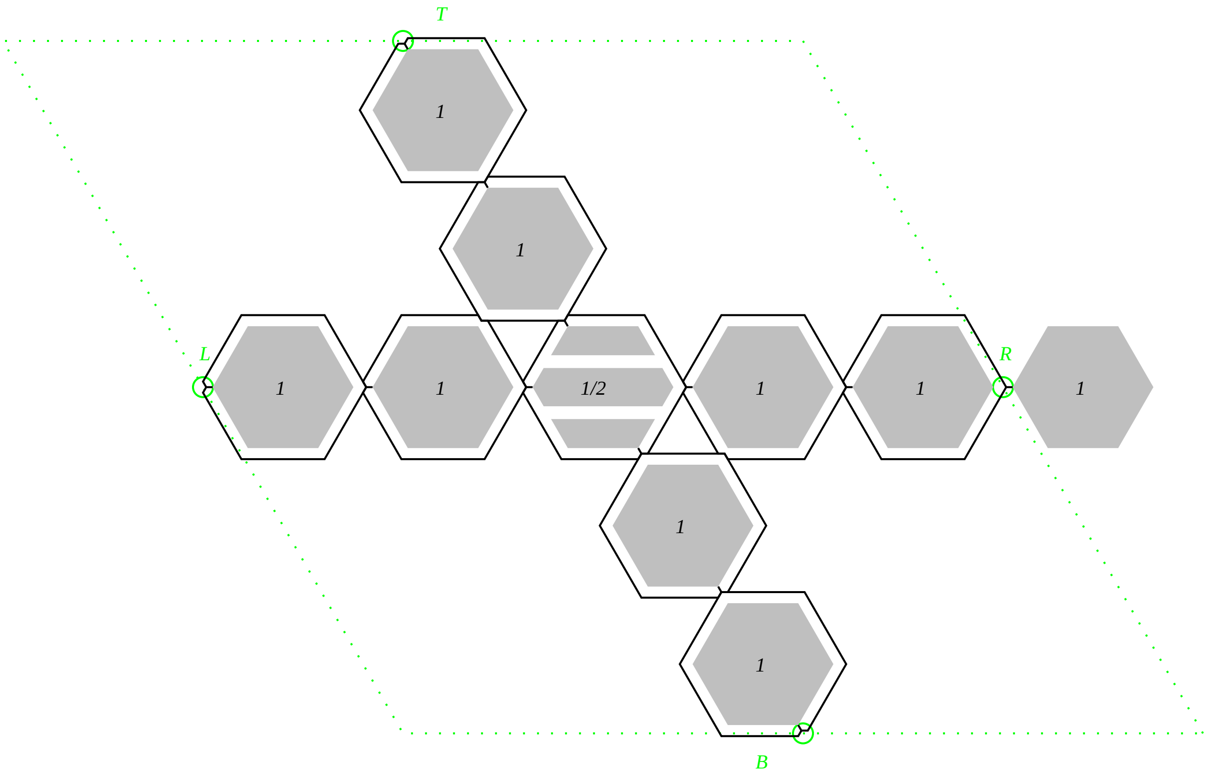


Figure 3.35: The positive literal component. The L and R terminals are drawn oriented East, the T terminal is drawn oriented South, and the B terminal is drawn oriented North.

drawn containing two quarter flippers and a half flipper. This component's key property is that for any realization, if the R terminal is oriented West, so that the terminal's flipper is in the literal's cage, then a full flipper must also be in the central cage, which means that all other terminals must be directed away from the component. A symmetric property holds for the negative literal component, shown in Figure 3.36. Note that the (unlabelled) positive and negative literal subgraphs are isomorphic. This is easier to see if you rotate one of the diagrams by 180 degrees.

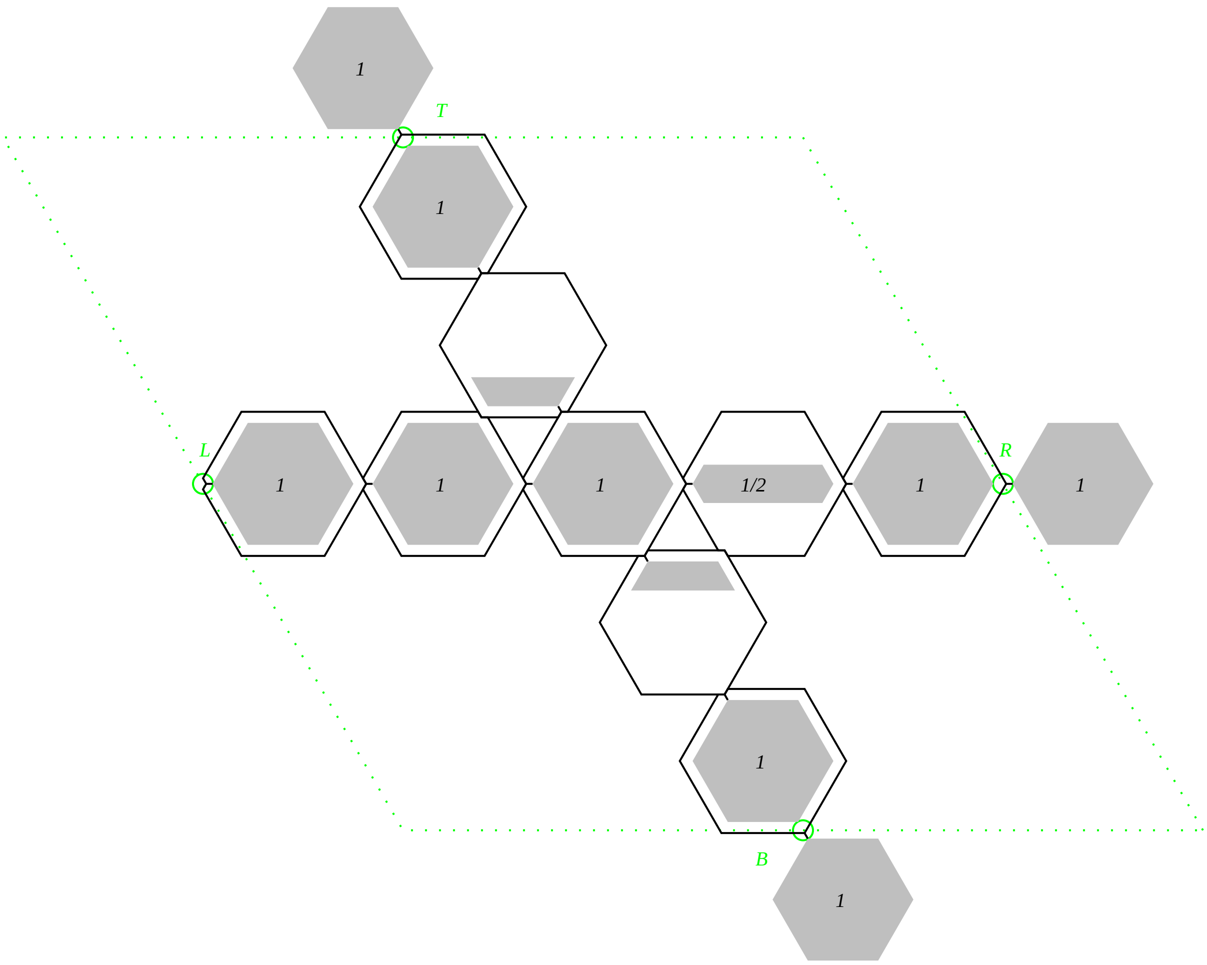


Figure 3.36: The negative literal component. The L and R terminals are drawn oriented East, the T terminal is drawn oriented North, and the B terminal is drawn oriented South.

Clauses

The clause testing component is shown in Figure 3.37. Its heart is again the central cage,

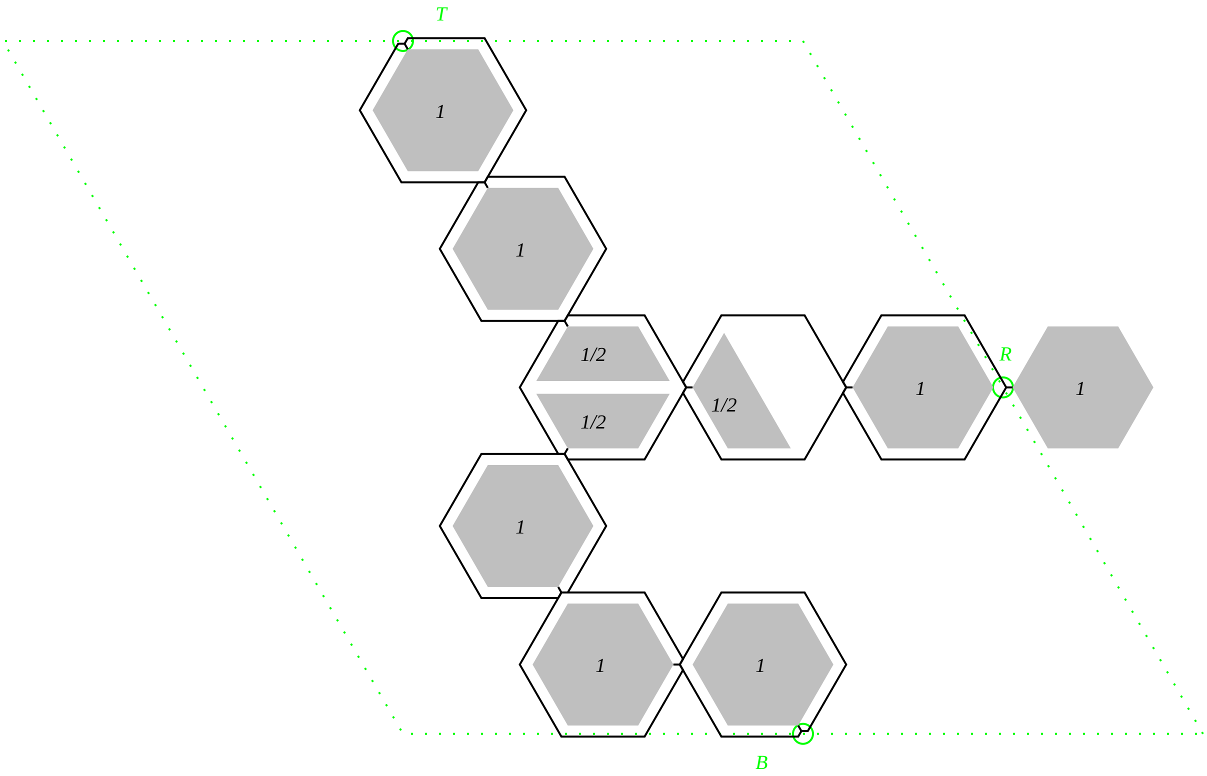


Figure 3.37: Clause component: drawn with the T terminal oriented South, the B terminal oriented North, and the R terminal oriented East.

the one containing two half flippers in the figure. Since this cage can contain at most two half flippers by Corollary 3.40.1, it must be that at least one terminal is oriented away from it.

If a terminal is not used in the grid drawing, that is, if there is no adjacent wire, corner, or crossover, then "cap" the corresponding graph component terminal by adding a small ring of cages with full flippers. The T terminal in Figure 3.38 is capped in this way. This cap ensures that one of the full flippers from the ring occupies the terminal's cage. This full cage acts as if the terminal were oriented towards the component, thereby

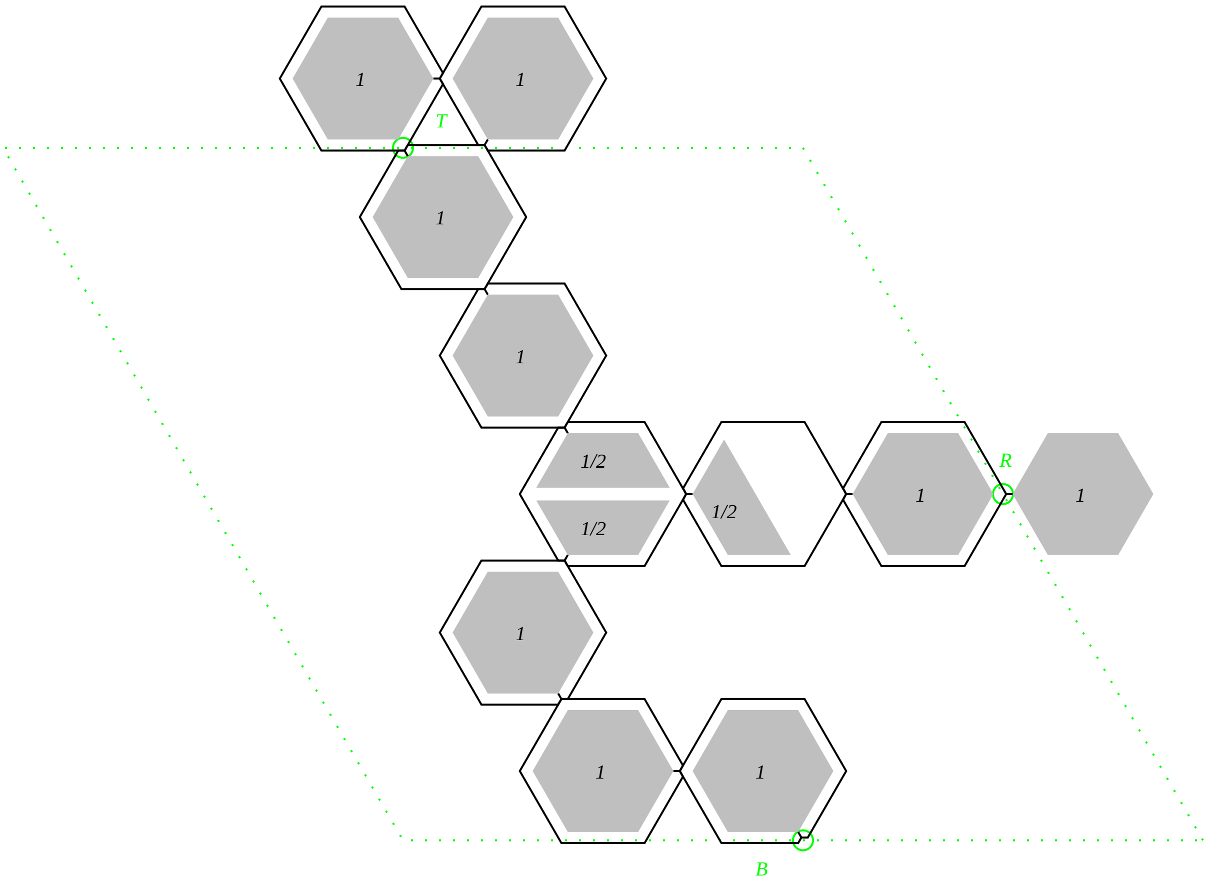


Figure 3.38: Capping a terminal.

forcing one of the remaining terminals to be oriented away[7].

3.4.9 Building and Realizing G_C

Now that all components have been described, we can present an example showing how components are connected. Recall that to connect two components, we simply identify the terminals at the desired connection point. Figure 3.39 shows how components are connected to simulate the skewed grid drawing from Figure 3.22. Coincidentally, the drawing in Figure 3.39 is oriented the same way as Figure 3.22, but remember that the object under construction is a *graph*, not a realization. To reiterate: we used realizations to specify graph components, but we need not appeal to realizations at all when connecting components.

Lemma 3.42 *The graph G_C has a realization if and only if the underlying grid drawing is orientable.*

Proof: Given a realization, orient the grid drawing terminals exactly as the realization terminals. The definition of orientation for realization terminals ensures that conditions **draw1** and **draw2** are met. The nature of the wire, corner, crossover, and clause components ensures that condition **draw3** is met. Finally, the nature of the literal components ensures that condition **draw4** is met. That is, the grid drawing is orientable if G_C has a realization.

Now assume that the terminals of the grid drawing have been oriented. From this we will construct a realization for G_C. For each component in the grid drawing, construct the corresponding component realization, oriented exactly as the grid drawing component, centered at the origin. The discussion accompanying the description of each graph

[7]Note that we could have simply "pinned" the unused flipper into its cage by dimpling in all corners and bending the cage such that all corners contact the flipper. It may not be possible to bend the cage for other potential uses of this reduction, for example, if we further restricted all cage disks to lie on the hexagonal grid.

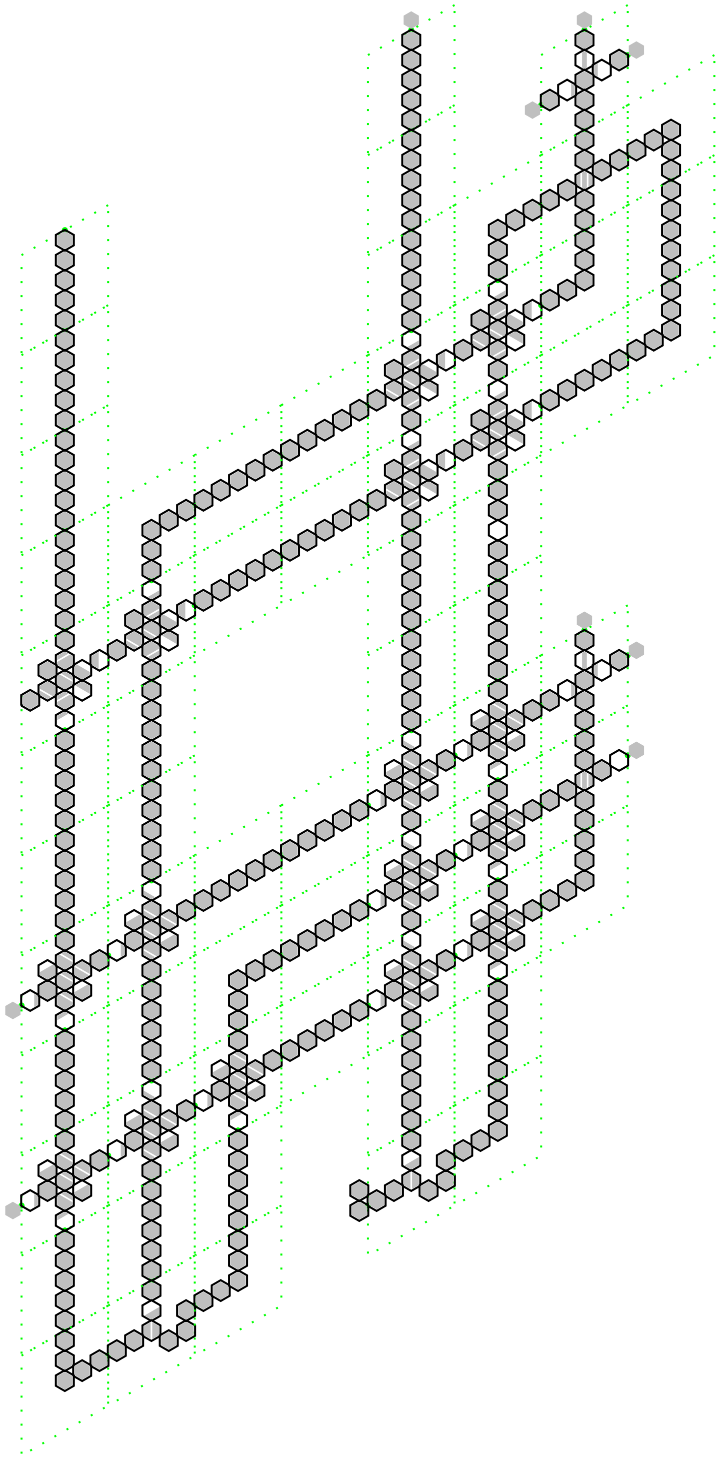


Figure 3.39: How to connect components. This “landscaped” figure (view it from the right) shows how to simulate the portion of the skewed grid drawing shown in Figure 3.22. There is a wire with both terminals directed outwards in Figure 3.21 and Figure 3.22. Do you see the empty cage corresponding to this wire?

component above implies that this can always be done. Now magnify the skewed grid such that the distance between grid coordinates is five hexagons. Then translate each oriented component realization to its grid location.

This procedure results in duplicated disks (points) corresponding to terminals. Remove one set; they were only duplicated for clarity of exposition of each component. ∎

Lemma 3.43 *SATISFIABILITY can be reduced to ρ-BOUNDED COIN GRAPH RECOGNITION in polynomial time, for any fixed ρ.*

Proof: The grid drawing has area $(6|U|+1)\times(3|C|+2)$, and therefore at most this many components, since a unit square encloses each component. Each component of G_C has a small constant number of cages and flippers. Each cage and flipper, on the other hand, has some, possible large, number of vertices and edges. But this number depends only on the ratio of disk diameters ρ, and not on the size of the SATISFIABILITY instance. Since the size of the SATISFIABILITY instance is a polynomial function of $|U|$ and $|C|$, the entire recognition instance can be built in polynomial time. ∎

3.4.10 Extending Coin Graph Recognition

Disk Intersection Graphs

We can easily modify the reduction for ρ-BOUNDED COIN GRAPH RECOGNITION to ρ-BOUNDED DISK GRAPH RECOGNITION as follows. First note that the cages are already suitable disk intersection graphs in that their realizations cannot contain more area by virtue of their disks having the freedom to overlap. The flippers pose only slightly more difficulty. Clearly, the realizations of current flippers may occupy *less* area by allowing their disks to overlap, thereby possibly leaving room for more flippers in a cage than we would like. The solution is to make the flippers out of *independent* vertices.

In this way they cannot be compressed smaller than (or even equal to) their packing size without coming into contact with one another. In their current construction, the flippers do not contact each other nor the cage (except at one attachment point). Therefore we have room to uniformly scale each flipper slightly while leaving the contact point fixed. This expansion disconnects the constituent disks from one another, as Figure 3.40.(a) shows in exaggerated proportions.

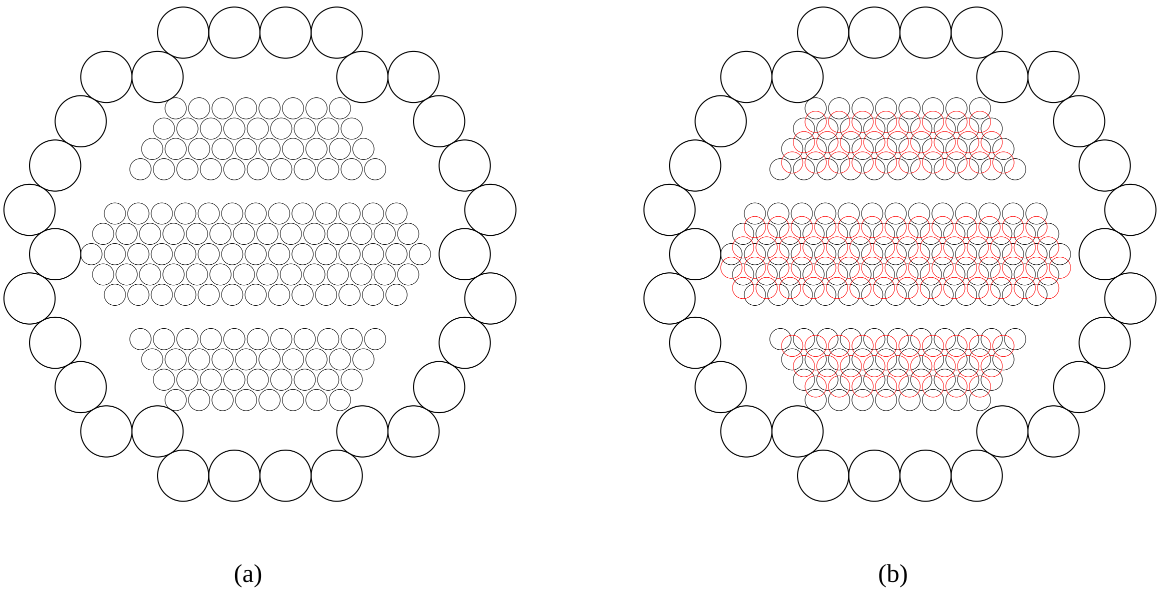


Figure 3.40: Expanding flippers to make the vertices independent. The flippers in the cage on the left have been expanded to make the vertices independent. The expanded flippers in the cage on the right have been "bound together" with the addition of many new disks. Although these new disks have been drawn as an independent set, there is no need for them to be independent.

Clearly, the number of disks in such an expanded flipper still approaches its asymptotic portion of the cage's capacity. We still need to hold the flipper together somehow, to ensure that it is completely embedded in some cage. We can easily accomplish this task by adding as many more disks to the flipper as we like, being careful not to intersect the cage nor any other flippers in the construction, as shown in Figure 3.40.(b). This modified reduction proves the following theorem.

Theorem 3.44 *ρ-BOUNDED DISK GRAPH RECOGNITION is* **NP***-hard for any fixed value $\rho \geq 1$.*

Corollary 3.44.1 *UNIT DISK GRAPH RECOGNITION is* **NP***-hard.*

Manhattan and Chessboard Metrics

Disks are bounded by squares under the Manhattan (L_1) and the chessboard (L_∞) metrics. We can easily modify the reduction for ρ-BOUNDED COIN GRAPH RECOGNITION (and the subsequent relaxation to arbitrary intersection) to ρ-BOUNDED SQUARE GRAPH RECOGNITION by skewing the grid differently and using cages that are diamonds rather than hexagons. To illustrate, assume that the squares are aligned with the coordinate axes. The new reduction is identical to ρ-BOUNDED COIN GRAPH RECOGNITION up to the point that the reduction draws the bipartite graph on the square grid.

Cages are diamonds, and are still joined at the corners by shared vertex triples, as shown in Figure 3.41.

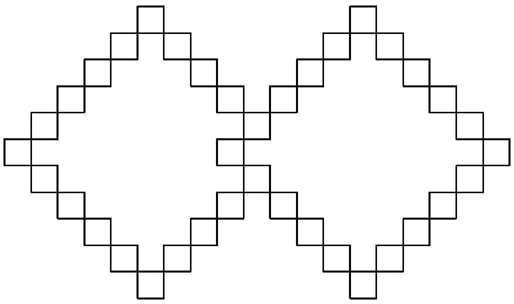


Figure 3.41: Joining square cages. Compare with Figure 3.25.

The unit L_∞ disks (squares) tile the plane in a square packing; this packing also constructs solid flippers leaving no internal gaps. The components are necessarily constructed differently, but the same principles apply. The modified reduction establishes the following theorem.

Theorem 3.45 *ρ-BOUNDED COIN GRAPH RECOGNITION is* **NP**-*hard for any fixed value $\rho \geq 1$ under the L_1, L_2, and L_∞ metrics.*

The extension to disk intersection graphs used for Theorem 3.44 applies here also.

Theorem 3.46 *ρ-BOUNDED DISK GRAPH RECOGNITION is* **NP**-*hard for any fixed value $\rho \geq 1$ under the L_1, L_2, and L_∞ metrics.*

Corollary 3.46.1 *ρ-BOUNDED SQUARE GRAPH RECOGNITION is* **NP**-*hard for any fixed value $\rho \geq 1$.*

Corollary 3.46.2 *UNIT SQUARE GRAPH RECOGNITION is* **NP**-*hard.*

Therefore, the smallest dimension K, for which a graph G is the intersection graph of K-dimensional unit hypercubes is called the *cubicity* of the graph [Rob68b]. This definition leads immediately to the following problem.

K-CUBICITY
INSTANCE: Graph $G = (V, E)$, positive integer K.
QUESTION: Does G have cubicity at most K?

The complexity of 2-CUBICITY was a long-standing open question [Rob68b, Coz92]. It follows from Corollary 3.46.2 that K-CUBICITY is **NP**-hard, even for $K = 2$. The Introduction mentions that graphs with cubicity 1 are called indifference graphs and can be recognized in polynomial time. That is, K-CUBICITY is solvable in polynomial time for $K = 1$. Also, Yannakakis [Yan82] showed that 3-CUBICITY is **NP**-hard.

Corollary 3.46.3 *2-CUBICITY is* **NP**-*hard.*

Conjecture 3.47 *K-CUBICITY is* **NP**-*hard for all fixed K greater than 1.*

Three Dimensions

The reduction can also be modified to three dimensions. A *k-dimensional ball* is the set of points within some radius of a point (the ball's centre) in k-dimensional Euclidean space. Hence disks are two-dimensional balls, and coins are two-dimensional interior-disjoint balls. The basic building blocks for the three-dimensional reduction are again flippers in cages, but this time the cages are three-dimensional regular octahedra. Again, derive an upper bound on the capacity of a cage by a spherical volume argument. Derive a lower bound on its octahedral capacity by packing the interior in a hexagonal lattice, which is a rotated face-centered cubic lattice. Note that it does not matter whether face-centered cubic is "optimal" (though it is the densest *lattice* packing in three dimensions [CS88]), merely that its capacity forms a small constant ratio with the spherical-volume upper bound.

Embed the SATISFIABILITY graph in the three-dimensional triangular grid as follows. Begin by embedding the clauses and variables in a two-dimensional square grid, as before. Think of this as a horizontal layer, with $z = 1$. Now, add more horizontal layers to the grid for a total of $3|C|$ layers, so that each occurrence of a literal in a clause has its own layer. Draw the bipartite graph corresponding to SATISFIABILITY on the three-dimensional grid by routing a literal to a clause. First route from the literal to the layer corresponding to the clause, using the third dimension. Then route over to the clause's (x, y) coordinates on its dedicated grid layer. Conclude by routing to the clause using the third dimension. In this way, no wires interfere so that the construction simulates a natural layout of the bipartite SATISFIABILITY graph without using cross-over components. Next, derive a tetrahedral three-dimensional grid by skewing the square grid. Finally, simulate the components with the octahedral cages. Note that these cages, unlike their two-dimensional analogues, have "holes in the mesh", which may allow flippers to "escape". Patch these holes with smaller spheres placed into the holes, possibly leaving some much smaller holes. Simply continue this process until all holes are smaller than the unit spheres.

Theorem 3.48 *In two or three dimensions, ρ-BOUNDED BALL GRAPH RECOGNITION and ρ-BOUNDED TOUCHING-BALL GRAPH RECOGNITION are* **NP**-*hard for any fixed ρ.*

Conjecture 3.49 *ρ-BOUNDED BALL GRAPH RECOGNITION and ρ-BOUNDED TOUCHING-BALL GRAPH RECOGNITION are* **NP**-*hard for all fixed ρ and for all fixed K greater than 1.*

Two unit balls intersect if and only if their boundaries, which are spheres, intersect. Therefore, the smallest dimension K, for which a graph G is the intersection graph of K-dimensional unit balls, is called the *sphericity* of the graph. This definition leads immediately to the following problem.

K-SPHERICITY

INSTANCE: Graph $G = (V, E)$, positive integer K.

QUESTION: Does G have sphericity at most K?

The complexity of 3-SPHERICITY was an open question due to Havel [Hav82a, Hav82b, HKC83] arising from studies of molecular conformation. It follows from Theorem 3.48 that K-SPHERICITY is **NP**-hard, even for $K = 2$ or $K = 3$. Again, recall from the Introduction that graphs with sphericity 1 are called indifference graphs and can be recognized in polynomial time. That is, K-SPHERICITY is solvable in polynomial time for $K = 1$.

Theorem 3.50 *K-SPHERICITY is* **NP**-*hard, even for $K = 2$ and $K = 3$.*

Conjecture 3.51 *K-SPHERICITY is* **NP**-*hard for all fixed K greater than 1.*

3.4.11 Miscellaneous Properties

The reduction embeds the bipartite graph corresponding to SATISFIABILITY in a very straightforward manner, making it easy for the reader to recall the structure of the drawing. This clarity comes at a small price, however, namely the need for crossover components. We could have constructed a reduction that avoids them. Kratochvíl [Kra94] has recently introduced a special case of SATISFIABILITY, called 4-BOUNDED PLANAR 3-CONNECTED 3-SAT, and shown that it is **NP**-complete. In this special case, every clause contains exactly three variables, every variable appears in at most four clauses, and the bipartite graph of clauses to *variables* is planar and 3-connected. We could have reduced this new problem to ρ-BOUNDED COIN GRAPHS, as follows. Since the bipartite graph is planar and has degree at most four, Valiant's procedure (see Section 3.3.1) will embed it in the square grid, with no crossovers, in polynomial time. The rest of the reduction would be nearly identical to the one presented here, differing only in that it does not require crossovers, and that the literal and clause components may have to have different shapes to accommodate the different incident edges.

We already noted that coin graphs are planar. On the other hand, no such restriction holds for disk graphs. However, note that an instance of the ρ-BOUNDED DISK GRAPH RECOGNITION constructed by the reduction in this chapter is planar. This is true whether or not it has a realization, since it is always possible to embed (in the traditional sense of drawing a graph on the plane) all flippers inside their incident cages without crossing edges by making them sufficiently small. That is, ρ-BOUNDED DISK GRAPH RECOGNITION remains **NP**-hard even if the graph is planar.

An instance of ρ-BOUNDED COIN GRAPH RECOGNITION constructed by the reduction has a 3-dimensional realization. To see this, embed the cages in the (horizontal) plane, as usual. Then embed each flipper directly above its incident vertex on the cage using the third dimension. Align all the flippers to ensure that flippers on a common cage remain independent of one another. That is, pick some plane normal to the horizontal and, when embedding a flipper, make all of its points equidistant from that plane. Since the cages are all spaced by the large hexagonal grid, the flippers from different cages will be, too, and will consequently be independent. This implies, for example, that 2-SPHERICITY remains **NP**-hard even if the graph has sphericity at most 3.

We can express COIN GRAPH RECOGNITION and DISK GRAPH RECOGNITION as existentially-quantified formulae in the first order theory of the reals (*cf.* [Can88]) as follows. Let $G = (V, E)$ be a graph where $V = \{1, 2, \ldots, n\}$. Say that $\{f_v : v \in V\} \subset \mathbf{R}^2$ is a set of (possible) disk locations. Similarly, say that $\{r_v : v \in V\} \subset \mathbf{R}$ is a set of (possible) disk radii. Then G is a coin graph if and only if

$$\begin{aligned}\exists f_1 \exists f_2 \ldots \exists f_n \exists r_1 \exists r_2 \ldots \exists r_n \quad & P_{coin}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) \\ & \wedge P_{\overline{E}}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n)\end{aligned}$$

where

$$P_{coin}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) = \bigwedge_{(u,v) \in E} \|f_u - f_v\| = r_u + r_v$$

and

$$P_{\overline{E}}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) = \bigwedge_{(u,v) \in \overline{E}} \|f_u - f_v\| > r_u + r_v. \tag{3.8}$$

Since the existential theory of the reals is decidable in **PSPACE** [Can88], it follows that COIN GRAPH RECOGNITION is in **PSPACE**[8].

[8]This should come as no surprise, since coin graphs can be recognized in linear time, as mentioned earlier.

We can add a predicate P_ρ to constrain the radii of the disks. In particular, given a value ρ, let

$$P_\rho(r_1, r_2, \ldots, r_n) = \bigwedge_{v \in V} 1 \le r_v \le \rho. \tag{3.9}$$

Then G is a ρ-bounded coin graph if and only if

$$\begin{aligned} \exists f_1 \exists f_2 \ldots \exists f_n \exists r_1 \exists r_2 \ldots \exists r_n \quad & P_{coin}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) \\ & \wedge P_{\overline{E}}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) \\ & \wedge P_\rho(r_1, r_2, \ldots, r_n). \end{aligned}$$

It follows that ρ-BOUNDED COIN GRAPH RECOGNITION is in **PSPACE**.

Similarly, G is a disk graph if and only if

$$\begin{aligned} \exists f_1 \exists f_2 \ldots \exists f_n \exists r_1 \exists r_2 \ldots \exists r_n \quad & P_{disk}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) \\ & \wedge P_{\overline{E}}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) \end{aligned}$$

where

$$P_{disk}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) = \bigwedge_{(u,v) \in E} \|f_u - f_v\| \le r_u + r_v$$

and $P_{\overline{E}}$ is defined by Equation 3.8. It follows that DISK GRAPH RECOGNITION is in **PSPACE**. Again, we can add predicate P_ρ (Equation 3.9). Then G is a ρ-bounded disk graph if and only if

$$\begin{aligned} \exists f_1 \exists f_2 \ldots \exists f_n \exists r_1 \exists r_2 \ldots \exists r_n \quad & P_{disk}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) \\ & \wedge P_{\overline{E}}(f_1, f_2, \ldots, f_n, r_1, r_2, \ldots, r_n) \\ & \wedge P_\rho(r_1, r_2, \ldots, r_n). \end{aligned}$$

It follows that ρ-BOUNDED DISK GRAPH RECOGNITION is in **PSPACE**.

Chapter 4

Cocomparability Graphs

Every τ-strip graph is a cocomparability graph for all $\tau \in [0, \sqrt{3}/2]$, as we established in Theorem 3.7. Therefore, the class of strip graphs naturally inherits the properties of, as well as any algorithms on, the class of cocomparability graphs. This chapter develops (in Section 4.2) polynomial time algorithms for several domination problems on cocomparability graphs. This chapter also establishes properties of cocomparability graphs that will be used later in the thesis to develop algorithms on strip graphs and two-level graphs. In particular, transitive orientations of the nonedges of strip graphs play a significant role in their characterization, as we will see in Chapter 5 and Chapter 6. Section 4.3 therefore explores the possible transitive orientations for the complements of cocomparability graphs.

The following brief introduction characterizes cocomparability graphs in terms of a linear order on the vertices of a graph. It also presents an algorithm for finding such an order. This characterization leads to efficient algorithms for several domination problems, which form the subject of Section 4.2. These algorithms, of course, are applicable to strip graphs and two-level graphs.

4.1 Introduction

Recall from Section 2.3 that a *comparability graph* is an undirected graph that has a transitive orientation, and that a graph is a *cocomparability graph* if its nonedges are transitively orientable (that is, if it is the complement of a comparability graph). Cocomparability graphs can be characterized in terms of possible linear orders on their vertices. In this chapter, we will see how this characterization leads to efficient algorithms.

Definition 4.1 A *spanning order* $(V, <)$ *of a graph* $G = (V, E)$ is a linear order on the vertices V of G such that, for any three vertices $u < v < w$, if u and w are adjacent, then v is adjacent to either u or w (or both). ∎

Figure 4.1 shows a small cocomparability graph and a spanning order on its vertices.

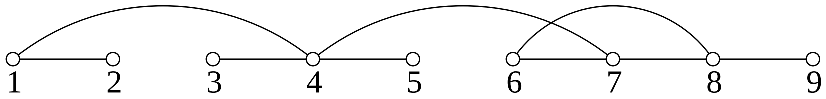


Figure 4.1: A cocomparability graph on nine vertices. The vertices are labelled in spanning order.

We will see that the vertices of every cocomparability graph can be labelled in spanning order (Lemma 4.2), and that every graph whose vertices can be labelled in spanning order is a cocomparability graph (Theorem 4.3). Damaschke [Dam92] says this is well-known, and he states the result without proof. Nevertheless, your intuition may be strengthened by working through the following simple proofs. In essence, Theorem 4.3 says that the restriction of a spanning order to the complementary edges is a transitive orientation.

Lemma 4.2 ([Dam92]) *Let* $G = (V, E)$ *be a cocomparability graph. Every linear extension of a transitive orientation of the complement of* G *is a spanning order of* G.

Proof: Let $G = (V, E)$ be a cocomparability graph. That is, its complement $\overline{G} = (V, \overline{E})$ is a comparability graph. Let $\vec{G} = (V, \vec{E})$ be a transitive orientation of $\overline{G}$, and consider any linear extension $(V, <)$ of $\vec{G}$. This is a spanning order of G. To see this, consider any three vertices $u < v < w$ such that $(u, w) \in E$ but $(u, v) \notin E$ and $(v, w) \notin E$. Then, by definition, $(u, v) \in \vec{E}$ and $(v, w) \in \vec{E}$. By transitivity, $(u, w) \in \vec{E}$, and therefore $(u, w) \notin E$, a contradiction. ∎

Theorem 4.3 ([Dam92]) *A graph is a cocomparability graph if and only if it has a spanning order.*

Proof: Every cocomparability graph has a spanning order by Lemma 4.2.

Conversely, suppose a graph $G = (V, E)$ has a spanning order. Construct a directed graph $\vec{G} = (V, \vec{E})$ such that $(u, v) \in \vec{E}$ if and only if $(u, v) \notin E$ and $u < v$. This graph is a transitive orientation of $\overline{G}$, for if $(u, v) \in \vec{E}$ and $(v, w) \in \vec{E}$, then $u < v < w$ and $(u, v) \notin E$ and $(v, w) \notin E$. Therefore u and w could not be adjacent; otherwise the spanning ordering property would be violated. That is, $(u, w) \in \vec{E}$. So $\overline{G}$ is a comparability graph, and G is a cocomparability graph. ∎

Definition 4.4 A *linearly-ordered graph* $G = (V, E, <)$ is a graph (V, E) and a linear order $(V, <)$. A *spanning-ordered graph* $G = (V, E, <)$ is a (necessarily cocomparability) graph (V, E) and a spanning order $(V, <)$. ∎

The algorithm below generates a spanning-ordered graph (several of the following sections require this) from a cocomparability graph. This algorithm may return a linearly-ordered graph even if G is not a cocomparability graph. This property allows us to use this linearly ordered graph, for example in the subsequent Steiner set algorithm (Section 4.2.5), to avoid having to recognize cocomparability graphs, a step that would add $\Theta(M(V))$ time [Spi85].

Theorem 4.5 *Given an arbitrary graph G, Algorithm OCC returns a spanning-ordered cocomparability graph if G is a cocomparability graph, or it returns a linearly-ordered*

Table 4.1: **Algorithm:** OCC(G) [spanning Ordered Cocomparability Graph]

Input: A graph $G = (V, E)$.
Output: A linearly-ordered graph, or
a message stating that G is not a cocomparability graph.

1 $\overline{G} \leftarrow$ the complement of G.
2 $\vec{G} \leftarrow$ TRANSITIVELY-ORIENT($\overline{G}$)
3 **if** the directed graph $\vec{G}$ does not contain a cycle,
4   **then** order the vertices of G by topologically sorting $\vec{G}$.
5   **return** the now linearly-ordered graph $G = (V, E, <)$.
6   **else** **return** "G is not a cocomparability graph."

graph, or it prints a message stating that G is not a cocomparability graph, in $O(V^2)$ time.

Proof: Implement Step 2 with Spinrad's transitive orientation algorithm [Spi85], which will orient any undirected graph. The resulting directed graph is transitive if and only if $\overline{G}$ is a cocomparability graph [Spi85]. This graph is clearly not transitive if it contains a cycle; this observation justifies Step 3. If G is a cocomparability graph, the linearly-ordered graph in Step 5 is a spanning-ordered graph by Lemma 4.2.

It is straightforward to implement Step 1 to run in $O(V^2)$ time. Spinrad's algorithm also takes $O(V^2)$ time [Spi85]. Step 4 can be implemented to run in $O(V + E)$ time ([CLR90] pages 485–488) using depth first search. Step 4 can simultaneously test for cycles in $\vec{G}$, since such a cycle exists if and only if depth first search discovers a "back edge" (one directed to an already-visited vertex). This obviates the need for an explicit Step 3. ∎

A graph is said to be *chordal* if it does not contain any induced cycles with four or more edges, that is, every such cycle has a "chord". Neither comparability graphs nor cocomparability graphs are necessarily chordal. In fact, the chordal cocomparability graphs are exactly the interval graphs [GH64]. However, cocomparability graphs are in some sense almost chordal. This is a simple consequence of spanning orders.

Theorem 4.6 ([Gal67]) *Cocomparability graphs do not have induced cycles with five or more edges.*

Proof: Let $G = (V, E)$ be a cocomparability graph, and let C be any cycle with five or more edges, and therefore at least five vertices, in G. Consider now a spanning order $(V, <)$, which must exist for G by Theorem 4.3. Let s be the least vertex in C in this order, and t the greatest vertex in C. Then C forms two paths from s to t, as shown in Figure 4.2. There must be three vertices $a < b < c$ such that b lies on one path, (a, c)

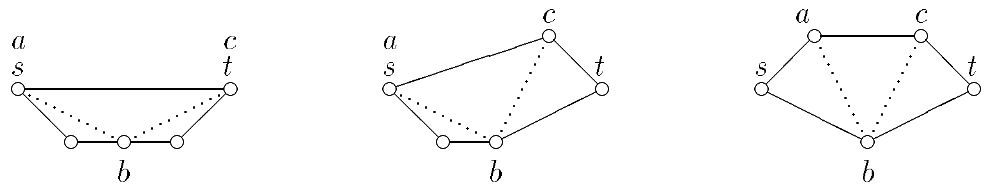


Figure 4.2: Every 5-cycle in a cocomparability graph has a chord: three cases in left-to-right order.

is an edge on the other path, and b is not adjacent to a or c (that is, $(a, b) \notin E$ and $(b, c) \notin E$). This contradicts the properties of a spanning order. ∎

A related idea is that of a dominating path. A path $P = (s, \ldots, t)$ in a graph $G = (V, E)$ is *dominating* if every graph vertex is either equal or adjacent to a path vertex.

Lemma 4.7 *A path P in a spanning-ordered cocomparability graph dominates all vertices between the least vertex in P and the greatest vertex in P.*

Proof: A path in any graph dominates the vertices on the path. So let v be a vertex in $G = (V, E, <)$, not in the path P, between the least vertex l and the greatest vertex t in P (that is, $l < v < t$). Then there exists some edge (u, w) of the path such that $l \leq u < v < w \leq t$. By the properties of the spanning order, v is adjacent to either u or w, so the path dominates v. ∎

Corollary 4.7.1 *A path in a spanning-ordered cocomparability graph dominates all vertices between its endpoints.*

4.2 Algorithms for Dominating and Steiner Set Problems

This section describes polynomial time algorithms for finding optimal dominating sets and Steiner sets in cocomparability graphs. The following table of contents lists the subsections in which each algorithm is developed.

An algorithm for this set is presented	in Subsection
minimum cardinality connected dominating set (MCCDS)	§4.2.1
minimum cardinality dominating set (MCDS)	§4.2.2
minimum cardinality total dominating set (MCTDS)	§4.2.3
minimum weight independent dominating set (MWIDS)	§4.2.4
minimum weight Steiner set (MWSS)	§4.2.5

Until recently [KS93], it was not known if these problems on cocomparability graphs were solvable in polynomial time. Every algorithm in this section has a better run time complexity, and is designed using a very different method, than the corresponding dynamic programming algorithm of Kratsch and Stewart [KS93]. Subsequently, Daniel Liang [Lia94] has achieved the same run time complexities as the algorithms in this section; his algorithms use dynamic programming and are similar to those of Kratsch and Stewart. A joint paper with Liang is in preparation.

Table 4.2 summarizes the results of this section and compares the running times of algorithms for interval graphs and permutation graphs, both of which are subclasses of cocomparability graphs. Recall that interval graphs are exactly the chordal cocomparability graphs. Dominating sets for interval graphs have received considerable attention [RR88b] and, in fact, linear algorithms for several dominating set problems are known. These algorithms, however, also exploit additional properties of interval graphs, such as chordality. The complexity of these problems on other kinds of perfect graphs appear in the literature and are succinctly surveyed by Corneil and Stewart [CS90b]. A particularly relevant observation from this survey is that all of these problems are **NP**-complete for comparability graphs, the complements of cocomparability graphs.

Table 4.2 assumes that a spanning order is available for the cocomparability graph algorithms, and that a defining permutation is available for the permutation graph algorithms. In both cases, these can be created in $O(V^2)$ time (see [Spi85] and Algorithm OCC in Table 4.1), but notice that doing so would introduce an $O(V^2)$ term to some of these algorithms. Incidentally, we may assume that our input graphs are connected, otherwise we can run the algorithms on the connected components and take the union of the solutions. In particular, this means $|V| = O(E)$.

Problem \ Graph	Cocomparability	Permutation	Interval
MCCDS	$O(VE)$	$O(V+E)$ [AR92]	$O(V+E)$ [RR88b]
MCDS	$O(VE^2)$	$O(V \log\log V)$ [TH90]	$O(V+E)$ [RR88b]
MCTDS	$O(VE^2)$	$O(V^2+VE)$ [CS90b]	$O(V+E)$ [RR88b]
MWIDS	$O(V^{2.376})$	$O(V^2)$ [BK87]	$O(V+E)$ [RR88b]
MWSS	$O(V \log V + E)$	$O(V \log V + E)$ [AR92]	

Table 4.2: Complexity of Domination Problems

Section 4.2.1 covers connected domination. Section 4.2.2 then develops an algorithm for finding a dominating set by exploiting the structure of a minimum cardinality connected dominating set. Subsequently, Section 4.2.3 reduces the problem of finding a total dominating set to that of finding a dominating set.

The algorithm for finding a minimum weighted independent dominating set utilizes a new $O(M(V))$ time algorithm for finding a minimum (or maximum) weight maximal clique in a comparability graph. Both algorithms are presented in Section 4.2.4.

The Steiner set algorithm MWSS-CC presented in Section 4.2.5 is particularly flexible: given an arbitrary graph, the algorithm will either find a minimum weight Steiner set, or it will print a message stating that the graph is not a cocomparability graph. That is, the $\Theta(M(V))$ time recognition step is not required.

Section 4.2.6 is on the applicability of these algorithms to realizations for three well known subclasses of cocomparability graphs. It shows that the standard realizations of permutation, interval, and indifference graphs admit easily extracted spanning orders, and that the algorithms therefore run unchanged on these representations.

4.2.1 Connected Dominating Sets

This section describes a polynomial time algorithm for finding a minimum cardinality connected dominating set (MCCDS) in a cocomparability graph. Recall that a connected dominating set in a graph is a subset of vertices that induces a connected subgraph, and that is adjacent to every other vertex in the graph.

If a vertex in a linearly-ordered graph dominates all lesser vertices (in the linear order), refer to it as a *left dominator*. Similarly, refer to a vertex that dominates all greater vertices as a *right dominator*. Let $V^-(G)$ denote the set of left dominators in the linearly-ordered graph G. Similarly let $V^+(G)$ denote the set of right dominators. We will not write the graph argument if it is implicit (that is, we write $V^+(G) = V^+$ if G is understood). Also let $V^-(S)$ (respectively $V^+(S)$) denote the set of left dominators (respectively right dominators) in the linearly-ordered graph induced by the linearly-ordered subset S of vertices V. More formally,

$$\begin{aligned} V^-(G) &= \{v : (u,v) \in E(G) \text{ for all vertices } u < v\}, \text{ and} \\ V^+(G) &= \{v : (u,v) \in E(G) \text{ for all vertices } u > v\}. \end{aligned}$$

Corollary 4.7.1 implies that every path from a vertex in V^- to a vertex in V^+ forms a connected dominating set. The following lemma implies that such a path could serve as an MCCDS if an MCCDS does not have exactly two vertices. In the forthcoming material, let $|P|$ denote the number of vertices in the set P, even if P is a path or similar structure.

Lemma 4.8 *Let M be a connected dominating set in a spanning-ordered cocomparability graph G. Then there exists a path P from V^- to V^+ such that (i) $|P| \leq |M|$, or (ii) $|P| = 3$ and $|M| = 2$.*

Proof:

Case 1: M contains vertices $u \in V^-$ and $w \in V^+$.

Since M induces a connected subgraph, there is a path P from u to w using only vertices in M. Therefore $|P| \leq |M|$.

Case 2: M does not contain any vertices from V^- but contains a vertex $w \in V^+$.

Let u_1 be the least (in the spanning order) vertex in V. Trivially, $u_1 \in V^-$ and so $u_1 \notin M$. Since M is a dominating set, u_1 is adjacent to some vertex $v_1 \in M$. See Figure 4.3.

Since $v_1 \notin V^-$, there is a least vertex $u_2 < v_1$ that is not adjacent to v_1. It must be that $u_2 \in V^-$ since every vertex $u < u_2$ is adjacent to u_2. To see this, note that such a vertex u is adjacent to v_1 since u_2 is the *least* vertex not adjacent to v_1. Then $u < u_2 < v_1$ implies that u_2 is adjacent to u according to the spanning order.

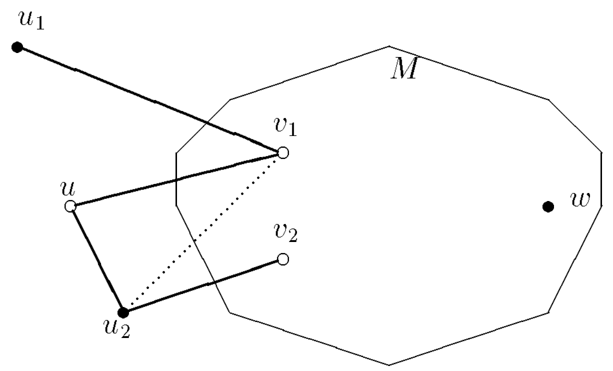


Figure 4.3: M does not contain any vertices from V^- but contains a vertex $w \in V^+$.

Again, u_2 is adjacent to some vertex $v_2 \in M$. Note that $v_2 \neq v_1$ since u_2 is not adjacent to v_1. Without loss of generality, suppose that $P' = (v_1, \ldots, w)$ is the shortest path from either v_1 or v_2 to w in the subgraph induced by M (otherwise relabel the vertices by exchanging subscript labels 1 and 2). This path exists since M is connected. The path cannot contain v_2 for, if it did, the path from v_2 would be shorter. Therefore $|P'| \leq |M \setminus \{v_2\}| = |M| - 1$. We can now create path P by appending u_1 to the beginning of P'. It follows that $|P| = |P'| + 1 \leq |M|$.

Case 3: M contains a vertex $w \in V^-$ but does not contain any vertices from V^+.

This proof is symmetric to that for Case 2. Since M does not contain any vertices from V^+, we can find distinct vertices $u_3 \in V^+$ and $u_4 \in V^+$ that are adjacent to distinct vertices v_3 and v_4 in M respectively (see Figure 4.4). We can now construct a shortest path P from w to u_3 or u_4 so that, again, $|P| \leq |M|$.

Case 4: M contains vertices from neither V^- nor V^+.

As for Case 2, we can find distinct vertices $u_1 \in V^-$ and $u_2 \in V^-$ that are adjacent to distinct vertices v_1 and v_2 in M respectively (see Figure 4.5). And as for Case 3, we can find distinct vertices $u_3 \in V^+$ and $u_4 \in V^+$ that are adjacent to distinct vertices v_3 and v_4 in M respectively (see Figure 4.5). Without loss of generality, let $P' = (v_1, \ldots, v_3)$ be

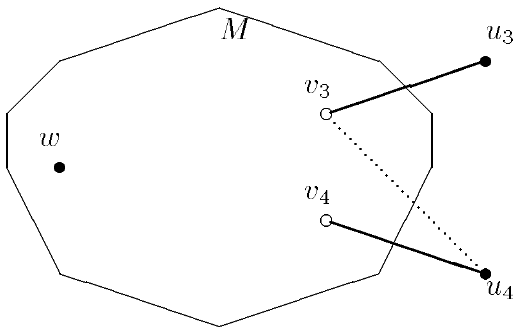


Figure 4.4: M contains a vertex $w \in V^-$ but does not contain any vertices from V^+.

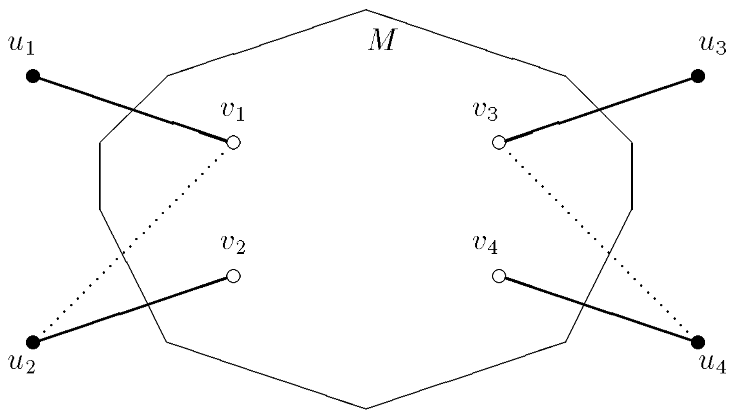


Figure 4.5: M contains vertices from neither V^- nor V^+.

the shortest path, from either v_1 or v_2 to either v_3 or v_4, in the subgraph induced by M (otherwise relabel the vertices). As before, this path cannot contain v_2, and it cannot contain v_4 either.

If $v_1 = v_3$, then $P = (u_1, v_1 = v_3, u_3)$ is a path in G that dominates G by the dominating properties of u_1 and u_3 and by Lemma 4.7. Then $|P| = 3$. It cannot be that $|M| = 1$, since otherwise the sole vertex would be in both V^- and V^+. Therefore either $|P| \leq |M|$ or $|M| = 2$.

On the other hand, if $v_1 \neq v_3$, then $v_2 \neq v_4$, otherwise $P' = (v_2)$ would have been the shortest path. Therefore $|P'| \leq |M \setminus \{v_2, v_4\}| = |M| - 2$. We can now create path P by appending u_1 to the beginning of P', and u_3 to the end of P'. It follows that $|P| = |P'| + 2 \leq |M|$. ∎

Figure 4.6 shows a graph with $|P| = 3$ and $|M| = 2$. The left-to-right order of the

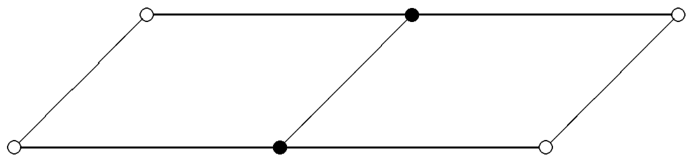


Figure 4.6: Lemma 4.8.(ii) A graph with $|P| = 3$ and $|M| = 2$.

vertices is clearly a spanning order. The two dark vertices form a connected dominating set, yet the shortest path from V^- to V^+ has three vertices.

Theorem 4.9 *Let G be a connected spanning-ordered cocomparability graph, and let P be a shortest path from V^- to V^+. Either P is a minimum cardinality connected dominating set, or $|P| = 3$ and G is dominated by a single edge.*

Proof: Let P be a shortest path from V^- to V^+. The path P is clearly connected, and it is dominating by the definitions of V^- and V^+, and by Lemma 4.7.

Now let M be a minimum cardinality connected dominating set of G. Then by Lemma 4.8, either $|P| \leq |M|$, and therefore P is a minimum cardinality connected dominating set, or $|P| = 3$ and $|M| = 2$, so that G is dominated by a single edge. ∎

Table 4.3: **Algorithm:** MCCDS-OCC(G) [Minimum Cardinality Connected Dominating Set for spanning-Ordered Cocomparability Graphs]

Input: A connected spanning-ordered cocomparability graph $G = (V, E)$
Output: A minimum connected dominating set of G

```
1   V⁻ ← {v : v dominates all lesser vertices }
2   V⁺ ← {v : v dominates all greater vertices }
3   P ← the shortest path from V⁻ to V⁺ in G
4   if |P| = 3
5     then for all edges (u, v) ∈ E
6              do if every vertex w ∈ V is adjacent to u or v
7                    then P ← {u, v}
8                         break
9   return P.
```

Theorem 4.10 *Algorithm MCCDS-OCC computes a minimum cardinality connected dominating set of a connected spanning-ordered cocomparability graph $G = (V, E, <)$ in $O(VE)$ time.*

Proof: The correctness of the algorithm follows immediately from Theorem 4.9. To analyze the run-time complexity of the algorithm, assume that the input graph is represented by an adjacency list and adjacency matrix.

Step 1 can be executed in $O(V + E)$ time by making use of the linear-order and the adjacency matrix. To test if a vertex is a left-dominator, we check all lesser vertices in order, stopping immediately if a vertex is not adjacent (i.e., not dominated). In this way every edge is visited at most once, and each vertex examines at most one non-adjacent

Table 4.4: **Algorithm:** MCCDS-CC(G) [Minimum Cardinality Connected Dominating Set for Cocomparability Graphs]

Input: A cocomparability graph $G = (V, E)$.
Output: A minimum cardinality connected dominating set,
or a message stating that no connected dominating set exists.

```
1   if G is connected,
2     then G ← OCC(G)
3          return MCCDS-OCC( G )
4     else return "No connected dominating set exists."
```

vertex. Similarly, Step 2 can be executed in $O(V + E)$ time by checking all greater vertices. Step 3 can be implemented by adding a new "source" vertex s and a new "target" vertex t that is adjacent to the vertices in V^- and V^+ respectively. Step 3 can then be executed in $O(V+E)$ time by running breadth first search starting at s. Finally, Step 6 runs in $O(V)$ time for a total of $O(VE)$ time since it is called for each edge. ∎

Theorem 4.11 *Given a cocomparability graph G, Algorithm MCCDS-CC computes a minimum cardinality connected dominating set, or prints a message stating that no connected dominating set exists, in $O(VE)$ time.*

Proof: Step 4 is correct since there is clearly no connected dominating set if G is not connected. On the other hand, if G is connected, V itself forms a connected dominating set. Step 2 correctly orders the vertices V of G by Theorem 4.5, so the set returned by Step 3 is a minimum cardinality connected dominating set by Theorem 4.10.

Step 1 takes $O(V + E)$ using depth first search, Step 2 takes $O(V^2)$ time by Theorem 4.5, and Step 3 takes $O(VE)$ time by Theorem 4.10. If G is connected, then $|E| \geq |V| - 1$. Therefore, if the algorithm executes Steps 2 and 3, then it must be that $O(V^2) = O(VE)$. The total time taken is therefore $O(V + E + VE) = O(VE)$. ∎

4.2.2 Minimum Cardinality Dominating Sets

We now know how to find a minimum cardinality *connected* dominating set in a cocomparability graph. On the other hand, there may be an even smaller set that dominates the graph but that does not induce a connected subgraph. This is the ordinary dominating set problem—to find a (not necessarily connected) minimum cardinality dominating set (MCDS) in a graph—which we now address.

Let $G = (V, E, <)$ be a spanning-ordered graph. To simplify the treatment of special cases, this section assumes that V has been augmented with two additional vertices s and t. These vertices $s = 0$ and $t = |V| + 1$ are not adjacent to any vertices. We will find an MCDS in G by finding a shortest path in an auxiliary digraph $G' = (V', A')$, which we will refer to as a *d-auxiliary graph* There is a node in V' for each vertex in V, and two nodes in V' for each undirected edge in E. More precisely,

$$\begin{aligned} V' &= V \cup V_{in} \cup V_{out}, \text{ where} \\ V_{in} &= \{e_{in} : e \in E\} \text{ and} \\ V_{out} &= \{e_{out} : e \in E\}. \end{aligned}$$

The arc set A' depends only on the spanning-ordered graph G. However, the reader may find its presentation at this point somewhat mysterious, so we will instead present A' in conjunction with its properties. For now, let us just say that A' is the disjoint union of six sets of directed arcs

$$A' = A_0 \cup A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5.$$

These sets are defined by the forthcoming Equations 4.10 through 4.15.

Let M be a minimum cardinality dominating set of G. Write

$$M = \bigcup_{i=0}^{k+1} P_i,$$

where each induced subgraph $G(P_i)$ is a connected component of the induced subgraph $G(M)$. Recall that $\mathrm{N}(P_i) = \mathrm{N}(P_i, G)$ denotes the vertices dominated by P_i in G. Clearly, P_i is a minimum cardinality connected dominating set of the subgraph induced by $\mathrm{N}(P_i)$. By Theorem 4.9, we may assume that P_i is either a shortest path $(l_i, \ldots, r_i)$ from a left dominator $l_i \in V^-(\mathrm{N}(P_i))$ to a right dominator $r_i \in V^+(\mathrm{N}(P_i))$, or a single edge $P_i = (l_i, r_i)$. We may further assume that $l_i \leq r_i$ in the spanning order for all paths P_i. We can clearly do so if $P_i = (l_i = r_i)$ or if $P_i = (l_i, r_i)$. For longer paths $P_i = (l_i, \ldots, r_i)$, we must have $l_i < r_i$, otherwise l_i and r_i would be adjacent by the definition of left and right dominators, and (l_i, r_i) would be a shorter path than P_i.

Furthermore, these paths do not overlap in the spanning order. That is, no vertex $v \in P_i$ falls between the least and greatest vertices of any other path P_j. This is because such a vertex v would be dominated by P_j according to Lemma 4.7, contradicting the fact that $G(P_i)$ and $G(P_j)$ are connected components of $G(M)$. We may therefore assume that

$$l_0 \leq r_0 < l_1 \leq r_1 < \cdots < l_{k+1} \leq r_{k+1}$$

in the spanning order. Note that, since the vertices s and t are not adjacent to any vertices, $P_0 = (l_0 = r_0 = s)$ and $P_{k+1} = (l_{k+1} = r_{k+1} = t)$. Define a *paths dominator* to be a set M that satisfies all of the above properties. We have established the following property of spanning-ordered graphs.

Property 4.12 *Every spanning-ordered graph has a paths dominator.*

We are now ready to derive the arcs of the d-auxiliary digraph. In order to do so, however, we will need the following technical lemma that explains why all vertices between two paths are dominated by the endpoints of the paths.

Lemma 4.13 *Let v be a vertex between paths in a paths dominator M, that is, $r_i < v <$*

l_{i+1} *for some* i. *Then* v *is adjacent to at least one endpoint of at least one of the two bracketing paths. More precisely,*

1. *If neither* P_i *nor* P_{i+1} *is an edge, then* v *is adjacent to* r_i *or* l_{i+1}.
2. *If* P_{i+1} *is an edge but* P_i *is not, then* v *is adjacent to* r_i, l_{i+1}, *or* r_{i+1}.
3. *If* P_i *is an edge but* P_{i+1} *is not, then* v *is adjacent to* l_i, r_i, *or* l_{i+1}.
4. *If both* P_i *and* P_{i+1} *are edges, then* v *is adjacent to* l_i, r_i, l_{i+1}, *or* r_{i+1}.

Proof: Since M is a dominating set, the vertex v is adjacent to some vertex $u \in M$. If $u \in P_i$, then we are done since either $r_i \in V^+(\mathrm{N}(P_i))$ or $P_i = (l_i, r_i)$. Similarly, we are done if $u \in P_{i+1}$, since either $l_{i+1} \in V^-(\mathrm{N}(P_{i+1}))$ or $P_{i+1} = (l_{i+1}, r_{i+1})$.

Otherwise u is in some other path P_j so that $u < r_i$ and $(u, r_i) \in \overline{E}$, or $l_{i+1} < u$ and $(u, l_{i+1}) \in \overline{E}$, since P_i and P_j do not overlap in the spanning order. In the first case, where $u < r_i < v$, there is an edge between u and v, but $(u, r_i) \in \overline{E}$. Therefore $(r_i, v) \in E$ by the spanning order. Similarly, in the second case, $(l_{i+1}, v) \in E$. ∎

We want to define the arcs A' of the d-auxiliary graph G' so that for every paths dominator M, there is a path P' from s to t in G' with the same number of vertices as M. Essentially, we want P' to simulate the sequence

$$S = (s = l_0 = r_0, l_1, \ldots, r_1, \ldots, l_i, \ldots, r_i, \ldots, l_{k+1} = r_{k+1} = t),$$

where $(l_i, \ldots, r_i)$ is the path P_i. More precisely, let $\sigma : V' \to V$ denote the *significance* of a vertex in G', defined by

$$\sigma(v') = \begin{cases} v' & \text{if } v' \in V \\ u & \text{if } v' = e_{in}, \text{ where } e = (u, v) \text{ and } u < v \\ v & \text{if } v' = e_{out}, \text{ where } e = (u, v) \text{ and } u < v. \end{cases}$$

We want the significance of path $P' = (s = p'_1, p'_2, \ldots, p'_n)$ to be S, that is, we want

$$S = (\sigma(p'_1), \sigma(p'_2), \ldots, \sigma(p'_n) = t),$$

or more compactly $S = \sigma(P')$, by a traditional abuse of notation. To achieve this, we need an arc from u' to v' in G' if $\sigma(u')$ and $\sigma(v')$ are consecutive in S. We will use dominating properties to determine which nodes should be adjacent.

To this end, let u and v be two consecutive vertices in the sequence S. If u and v belong to some path P_i, then $(u, v) \in E$, and by the spanning order, $\{u, v\}$ dominates all vertices between u and v. Therefore define $A_1 \cup A_2$ to be the set of arcs whose endpoints dominate all vertices in between. The set A_2 is just the reversal of A_1. More precisely,

$$\begin{aligned} A_1 &= \{(u, v) : u, v \in V,\ u < v, \text{ and for all } i \in V, \\ &\qquad u < i < v \text{ implies } (i, u) \in E \text{ or } (i, v) \in E\} \qquad (4.10) \\ A_2 &= \{(v, u) : (u, v) \in A_1\}. \qquad (4.11) \end{aligned}$$

If u and v do not belong to a path, then $u = r_i$ and $v = l_{i+1}$ for some paths P_i and P_{i+1}. We must consider four cases, depending on whether P_i or P_{i+1} is an edge.

Case 1: Neither P_i nor P_{i+1} is an edge.

In this case, $\{r_i, l_{i+1}\}$ dominates all vertices in between by Lemma 4.13.1. Note that $(r_i, l_{i+1}) \in A_1$.

Case 2: $P_{i+1} = (l_{i+1}, r_{i+1}) = e$ is an edge but P_i is not.

The set $\{r_i\} \cup \{l_{i+1}, r_{i+1}\}$ dominates all vertices between r_i and l_{i+1} by Lemma 4.13.2. In this case, we want an arc from r_i to e_{in}. More precisely,

$$\begin{aligned} A_3 &= \{(w, e_{in}) : w \in V,\ e = (u, v) \in E,\ w < u < v, \text{ and for all } i \in V, \\ &\qquad w < i < u \text{ implies } (i, u) \in E,\ (i, v) \in E, \text{ or } (i, w) \in E\}. \qquad (4.12) \end{aligned}$$

Also, we wish to ensure that any path in G' through e_{in} also includes e_{out}. To do so, we define the arcs A_0 between such pairs. For every edge $e \in E$, this will be the only arc *from* e_{in} and the only arc *to* e_{out}. More precisely,

$$A_0 = \{(e_{in}, e_{out}) : e \in E\}. \tag{4.13}$$

Case 3: $P_i = (l_i, r_i) = e$ is an edge but P_{i+1} is not.

The set $\{l_i, r_i\} \cup \{l_{i+1}\}$ dominates all vertices between r_i and l_{i+1} by Lemma 4.13.3. In this case, we want an arc from e_{out} to l_{i+1}. More precisely,

$$\begin{aligned} A_4 \;=\; & \{(e_{out}, w) : w \in V,\ e = (u, v) \in E,\ u < v < w,\ \text{and for all } i \in V, \\ & v < i < w \text{ implies } (i, u) \in E,\ (i, v) \in E,\ \text{or } (i, w) \in E\}. \end{aligned} \tag{4.14}$$

Again, the arcs in A_0 ensure that any path in G' through e_{out} includes e_{in}.

Case 3: Both $P_i = (l_i, r_i) = e$ and $P_{i+1} = (l_{i+1}, r_{i+1}) = f$ are edges.

The set $\{l_i, r_i\} \cup \{l_{i+1}, r_{i+1}\}$ dominates all vertices between r_i and l_{i+1} by Lemma 4.13.4. In this case, we want an arc from e_{out} to f_{in}. More precisely,

$$\begin{aligned} A_5 \;=\; & \{(e_{out}, f_{in}) : e = (u, v),\ f = (w, x) \in E,\ u < v < w < x,\ \text{and for all } i \in V, \\ & v < i < w \text{ implies } (i, u) \in E,\ (i, v) \in E,\ (i, w) \in E,\ \text{or } (i, x) \in E\}. \end{aligned} \tag{4.15}$$

This completes our description of the arc sets A_0 through A_5 in A'. The preceding discussion establishes the following lemma.

Lemma 4.14 *For every paths dominator M of a spanning-ordered graph, there is a path P' in G' that satisfies $|P'| \leq |M|$.*

The reader may suspect from our construction that every path in G' also corresponds to a dominating set in G. The following lemma confirms this suspicion.

Lemma 4.15 *Let P' be a path from s to t in G'. The vertices of V corresponding to the nodes of P' dominate G.*

Proof: Let $S = \{\sigma(v) : v \in P'\}$ be the set of vertices corresponding to the nodes in the path P', and let j be an arbitrary vertex in V. We claim that S dominates j.

If $j \in S$, then we are done. Otherwise, $s < j < t$ in the spanning order, so there exists a pair of vertices i and k in P' such that $(i,k) \in A'$ and either $\sigma(i) < j < \sigma(k)$ or $\sigma(k) < j < \sigma(i)$.

If $(i,k) \in A_0$, then $(\sigma(i), \sigma(k)) \in E$ so j is dominated by $\sigma(i)$ or $\sigma(k)$ by the spanning order. If $(i,k) \in A_1$ or $(i,k) \in A_2$ then $(j, \sigma(i)) \in E$ or $(j, \sigma(k)) \in E$ by the definition of the sets.

If $(i,k) \in A_3$, then $(i,k) = (w, e_{in})$ for some vertex w and edge $e = (u,v)$ that satisfies $w < j < u < v$. Therefore j is adjacent to u, v, or w by the definition of A_3. Since node e_{in} has outdegree 1 in G', it must be that $e_{out} \in P'$. Therefore $v = \sigma(e_{out}) \in S$ and, since we already know that $u = \sigma(e_{in}) = \sigma(k)$ and $w = \sigma(i)$ are in S, it follows that j is dominated by S.

If $(i,k) \in A_4$, then $(i,k) = (e_{out}, w)$ for some vertex w and edge $e = (u,v)$ such that $u < v < j < w$. Therefore j is adjacent to u, v, or w by the definition of A_4. Since node e_{out} has indegree 1 in G', it must be that $e_{in} \in P'$. Therefore $u = \sigma(e_{in}) \in S$ and, since we already know that $v = \sigma(e_{out}) = \sigma(i)$ and $w = \sigma(k)$ are in S, it follows that j is dominated by S.

Finally, if $(i,k) \in A_5$, then $(i,k) = (e_{out}, f_{in})$ for some edges $e = (u,v)$ and $f = (w,x)$ such that $u < v < j < w < x$. Therefore j is adjacent to u, v, w, or x by the definition of A_5. Since node e_{out} has indegree 1 and node f_{in} has outdegree 1 in G', it must be that $e_{in} \in P'$ and $f_{out} \in P'$. Therefore $u = \sigma(e_{in}) \in S$ and $x = \sigma(f_{out}) \in S$ and, since we already know that $v = \sigma(e_{out}) = \sigma(i)$ and $w = \sigma(f_{in}) = \sigma(k)$ are in S, it follows that j is dominated by S. ∎

Theorem 4.16 *The set of vertices corresponding to the nodes in a shortest path from s*

to t in G' is a minimum cardinality dominating set.

Proof: Such a set is dominating by Lemma 4.15 and of minimum cardinality by Lemma 4.14. ∎

Let us now examine a straightforward algorithm for constructing the d-auxiliary digraph G'. To enhance the legibility of the algorithm, we will construct a subalgorithm for each of the arc sets A_0 through A_5. An implementation of this algorithm could combine the subalgorithms to capitalize on loops that are shared by more than one subalgorithm. The most straightforward of the subalgorithms is the one for A_0. It clearly executes in $O(E)$ time.

Table 4.5: **Algorithm:** A0-MCDS-OCC(G) [Construct arc set A_0]

Input: A spanning-ordered graph $G = (V, E, <)$.
Output: The arc set A_0 (Equation 4.13)

$A_0 \leftarrow \emptyset$
for all edges $e \in E$ where $e = (u, v)$ and $u < v$
  do $A_0 \leftarrow A_0 \cup \{(e_{in}, e_{out})\}$
return A_0

To construct the arcs for A_1, we only need to determine which pairs dominate all vertices in between. This is certainly true for the endpoints of edges in G. Otherwise, Algorithm A1-MCDS-OCC (Table 4.6) tests if a vertex is *not* dominated by the pair. Step 2 calls Step 4 $O(V^2)$ times and calls Step 5 $O(\overline{E})$ times. Step 5 calls Step 6 $O(V)$ times. Therefore Algorithm A1-MCDS-OCC runs in $O(V^2 + V\overline{E}) = O(V^3)$ time. Note that there are $O(V^2)$ arcs in A_1.

To construct the arcs for A_3, we need to determine which pairs of vertices and edges dominate everything in between. Again, Algorithm A3-MCDS-OCC (Table 4.7) tests if a vertex is *not* dominated by the pair. Step 2 calls Step 3 $O(E)$ times, which calls Step 5

Table 4.6: **Algorithm:** A1-MCDS-OCC(G) [Construct arc set A_1]

Input: A spanning-ordered graph $G = (V, E, <)$.
Output: The arc set A_1 (Equation 4.10)

$A_1 \leftarrow \emptyset$
for all pairs of vertices $u, v \in V$ where $u < v$
  do $A_1 \leftarrow A_1 \cup \{(u, v)\}$
    if $(u, v) \notin E$
      for all i such that $u < i < v$
        if $(i, u) \notin E$ and $(i, v) \notin E$
          then $A_1 \leftarrow A_1 \setminus \{(u, v)\}$
return A_1

$O(V)$ times, which calls Step 6 $O(V)$ times. Therefore Algorithm A3-MCDS-OCC runs in $O(V^2E)$ time. Note that there are $O(VE)$ arcs in A_3.

Table 4.7: **Algorithm:** A3-MCDS-OCC(G) [Construct arc set A_3]

Input: A spanning-ordered graph $G = (V, E, <)$.
Output: The arc set A_3 (Equation 4.12)

1 $A_3 \leftarrow \emptyset$
2 **for all** edges $e \in E$ where $e = (u, v)$ and $u < v$
3   **do for all** vertices w such that $w < u < v$
4   **do** $A_3 \leftarrow A_3 \cup \{(w, e_{in})\}$
5   **for all** vertices i such that $w < i < u$
6   **do if** $(i, w) \notin E$, $(i, u) \notin E$, and $(i, v) \notin E$
7   **then** $A_3 \leftarrow A_3 \setminus \{(w, e_{in})\}$
8 **return** A_3

Algorithm A4-MCDS-OCC (Table 4.8) is symmetric with Algorithm A3-MCDS-OCC. It therefore also runs in $O(V^2E)$ time, and A_4 has $O(VE)$ arcs.

In Algorithm A5-MCDS-OCC (Table 4.9), Step 2 calls Step 3 $O(E)$ times, which calls

Table 4.8: **Algorithm:** A4-MCDS-OCC(G) [Construct arc set A_4]

Input: A spanning-ordered graph $G = (V, E, <)$.
Output: The arc set A_4 (Equation 4.14)

1 $A_4 \leftarrow \emptyset$
2 **for all** edges $e \in E$ where $e = (u, v)$ and $u < v$
3   **do for all** vertices w such that $v < w$
4   **do** $A_4 \leftarrow A_4 \cup \{(e_{out}, w)\}$
5   **for all** vertices i such that $v < i < w$
6   **do if** $(i, u) \notin E$, $(i, v) \notin E$, and $(i, w) \notin E$
7   **then** $A_4 \leftarrow A_4 \setminus \{(e_{out}, w)\}$
8 **return** A_4

Step 5 $O(E)$ times, which calls the constant-time Step 6 step $O(V)$ times. Therefore Algorithm A5-MCDS-OCC executes in $O(VE^2)$ time. Note that there are $O(E^2)$ arcs in A_5.

Table 4.9: **Algorithm:** A5-MCDS-OCC(G) [Construct arc set A_5]

Input: A spanning-ordered graph $G = (V, E, <)$.
Output: The arc set A_5 (Equation 4.15)

1 $A_5 \leftarrow \emptyset$
2 **for all** edges $e \in E$ where $e = (u, v)$ and $u < v$
3   **do for all** edges $f \in E$ where $f = (w, x)$ and $u < v < w < x$
4   **do** $A_5 \leftarrow A_5 \cup \{(e_{out}, f_{in})\}$
5   **for all** vertices i such that $v < i < w$
6   **do if** $(i, u) \notin E$, $(i, v) \notin E$, $(i, w) \notin E$, and $(i, x) \notin E$
7   **then** $A_5 \leftarrow A_5 \setminus \{(e_{out}, f_{in})\}$
8 **return** A_5

Lemma 4.17 *Algorithm Aux-MCDS-OCC (Table 4.10) constructs the d-auxiliary graph in $O(VE^2)$ time.*

Table 4.10: **Algorithm:** Aux-MCDS-OCC(G) [d-Auxiliary Graph for Minimum Cardinality Dominating Set for spanning-Ordered Cocomparability Graphs]

Input: A spanning-ordered graph $G = (V, E, <)$.
Output: The d-auxiliary digraph $G' = (V', A')$.

1 $V' \leftarrow V \cup E_{in} \cup E_{out}$
2 $A_0 \leftarrow$ A0-MCDS-OCC(G)
3 $A_1 \leftarrow$ A1-MCDS-OCC(G)
4 $A_2 \leftarrow A_1^{-1}$
5 $A_3 \leftarrow$ A3-MCDS-OCC(G)
6 $A_4 \leftarrow$ A4-MCDS-OCC(G)
7 $A_5 \leftarrow$ A5-MCDS-OCC(G)
8 $A' \leftarrow A_0 \cup A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5$
9 **return** $G' = (V', A')$

Proof: It is straightforward to verify that the algorithm correctly constructs arc sets A_0 through A_5. Of course, the algorithm refers to these sets for clarity only. An implementation would just add the arcs directly to the set A'.

Implement the input and output graphs as adjacency matrices. Testing if two vertices are adjacent in G, or adding an arc to G' therefore take constant time. Step 1 amounts to allocating memory for an $O((V+2E)\times(V+2E))$ matrix, and initializing it in $O(V^2+E^2)$ time. It is easy to verify that Step 7 dominates the running time of the algorithm and takes at most $O(VE^2)$ time from the discussion above. ∎

Theorem 4.18 *Algorithm MCDS-OCC finds a minimum cardinality dominating set of a spanning-ordered cocomparability graph in $O(VE^2)$ time.*

Proof: The algorithm is correct by Theorem 4.16. Step 1 takes $O(VE^2)$ time by Lemma 4.17. Step 2 can be implemented to run in $O(V'^2) = O(V^2 + E^2)$ time using breadth first search[CLR90], and Step 3 clearly runs in $O(V)$ time. Step 1 dominates the run time of the algorithm, so the theorem follows. ∎

Table 4.11: **Algorithm:** MCDS-OCC(G) [Minimum Cardinality Dominating Set for spanning-Ordered Cocomparability Graphs]

Input: A spanning-ordered graph $G = (V, E, <)$.
Output: A minimum cardinality dominating set of G.

$G' \leftarrow$ MCDS-OCC-Aux(G)
$P' \leftarrow$ a shortest path from s to t in G'.
$S \leftarrow \{\sigma(v) : v \in P'\}$
return S.

4.2.3 Total Dominating Sets

Recall from the introduction (§1.3.2) that a *total dominating set* of a graph $G = (V, E)$ is a subset $D \subseteq V$ such that every vertex in V is adjacent to some vertex in D. Contrast this with an ordinary dominating set, which *either* contains *or* is adjacent to every graph vertex. The *total domination problem* is to find a total dominating set of minimum cardinality.

This section presents an efficient algorithm for total domination in cocomparability graphs. It does this by reducing the problem to (ordinary) domination in cocomparability graphs. Given an arbitrary graph G, we begin by constructing an (undirected) auxiliary graph G' that we will refer to as a *t-auxiliary graph*. We will see that if G is a cocomparability graph, then so is G'. Furthermore, there is a minimum cardinality dominating set M of G' that corresponds to a minimum cardinality *total* dominating set of G. That is, M solves the total domination problem on G.

Definition 4.19 Let $G = (V, E)$ be an arbitrary graph. The *t-auxiliary graph* G' is two copies of the graph G. In addition, two vertices in different copies are adjacent if the corresponding vertices in G are adjacent. More formally, G' is defined by the following equations.

$$
\begin{aligned}
G' &= (V', E') \text{ where} \\
V' &= V^1 \cup V^2 \\
E' &= E^1 \cup E^2 \cup E^{12} \\
V^1 &= \{v^1 : v \in V\} \\
V^2 &= \{v^2 : v \in V\} \\
E^1 &= \{(u^1, v^1) : (u, v) \in E\} \\
E^2 &= \{(u^2, v^2) : (u, v) \in E\} \\
E^{12} &= \{(u^1, v^2) : (u, v) \in E\}
\end{aligned}
$$

∎

To illustrate this definition, Figure 4.7 shows the t-auxiliary graph of the graph in Figure 4.1. For the rest of this subsection, define the *significance* $\sigma(v')$ of a vertex in V'

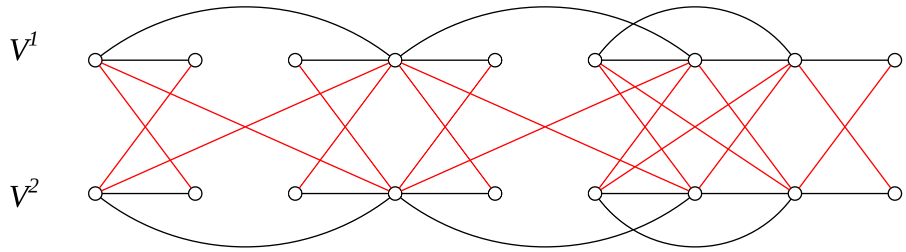


Figure 4.7: Reducing total domination to domination. This is the t-auxiliary graph of the cocomparability graph in Figure 4.1. The vertices V^1 are on the top, and the vertices v^2 are on the bottom. Similarly, the edges E^1 are on the top and the edges E^2 are on the bottom. The edges E^{12} go between the two layers of vertices.

to be the corresponding vertex in V. More formally, let $\sigma : V' \to V$ be a function that satisfies $\sigma(v^1) = \sigma(v^2) = v$ for all vertices $v \in V$.

We can now make three simple observations. Note that Observations 4.21 and 4.22 can be viewed as corollaries of Observation 4.20.

Observation 4.20 *Every pair of vertices $u', v' \in V'$ satisfies $(u', v') \in E'$ if and only if $(\sigma(u'), \sigma(v')) \in E$.*

Observation 4.21 *Vertices v^1 and v^2 are not adjacent in G' for every $v \in V$.*

Observation 4.22 *A vertex $w' \in G'$ is adjacent to v^1 if and only if w' is adjacent to v^2. That is, $\mathrm{Adj}(v^1, G') = \mathrm{Adj}(v^2, G')$ for every $v \in V$.*

Lemma 4.23 *The t-auxiliary graph of a cocomparability graph is also a cocomparability graph.*

Proof: Let $G = (V, E)$ be a cocomparability graph, and let $G' = (V', E')$ be its t-auxiliary graph using the notation in Definition 4.19. If $(V, <)$ is a spanning order of G, then the linear order $(V', \prec)$, where $u' \prec v'$ if and only if $\sigma(u') < \sigma(v')$, or $u' = v^1$ and $v' = v^2$ for some $v \in V$, is a spanning order of G'. To see this, let $a \prec b \prec c$ be three vertices in G' such that a and c are adjacent. Then $\sigma(a) \leq \sigma(b) \leq \sigma(c)$, and $(\sigma(a), \sigma(c)) \in E$ by Observation 4.20. Therefore, if $\sigma(b)$ is equal to one of $\sigma(a)$ or $\sigma(c)$, then it is adjacent to the other. On the other hand, if $\sigma(b)$ is distinct from $\sigma(a)$ and $\sigma(c)$, then $\sigma(b)$ is also adjacent to $\sigma(a)$ or $\sigma(c)$ by the spanning order on G. Therefore b is adjacent to a or c by Observation 4.20. If follows from Lemma 4.3 that G' is a cocomparability graph. ∎

Suppose now that $M^{12} \subseteq V^1 \cup V^2$ is a dominating set of G'. Then there is a subset $M^1 \subseteq V^1$ that is also a dominating set of G', and that satisfies $|M^1| \leq |M^{12}|$. We will see shortly (Lemma 4.27) that $\sigma(M^1)$ is a total dominating set for G. But first, we need to see how to construct M^1 from a dominating set M^{12}. We will do this by "lifting" vertices from $M^{12} \cap V^2$ into V^1. Let v^2 be an arbitrary vertex in $M^{12} \cap V^2$. If $v^1 \notin M^{12}$, then lift v^2 to v^1. Otherwise, $v^1 \in M^{12}$. Since G has no isolated vertices, v^1 must have a neighbour $w^1 \in V^1$. In this case, lift v^2 to vertex w^1 (any such neighbour will do). More formally, let $M^1 = f(M^{12})$ where $f : V' \to V^1$ is defined by the following equation.

$$f(v') = \begin{cases} v^1 & \text{if } v' = v^1 \\ v^1 & \text{if } v' = v^2 \text{ and } v^1 \notin M^{12} \\ w & \text{if } v' = v^2,\ v^1 \in M^{12}, \text{ and } (v^1, w^1) \in E^1 \end{cases} \tag{4.16}$$

For example, the circled vertices in Figure 4.8 form a dominating set (actually a

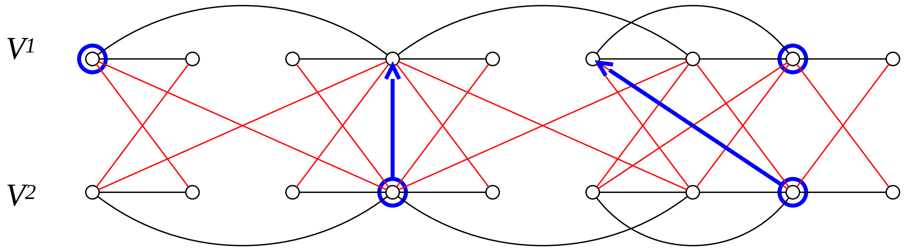


Figure 4.8: The effect of function f on a minimum cardinality dominating set.

minimum cardinality dominating set) of the graph in Figure 4.7. The arrows in Figure 4.8 show the effect of the function f. Since f maps vertices already in the set V^1 to themselves, the figure does not show arrows from (or to) these vertices.

Observation 4.24 *If $v^1 \in M^{12}$ or $v^2 \in M^{12}$, then $v^1 \in f(M^{12})$.*

Observation 4.25 $|M^1| \leq |M^{12}|$.

Proof: The observation follows since f (Equation 4.16) is a function. ▮

Lemma 4.26 *Let G' be the t-auxiliary graph of a cocomparability graph G with no isolated vertices. If M^{12} is a dominating set of G', then $M^1 = f(M^{12})$ is a dominating set of G'.*

Proof: Let v' be an arbitrary vertex in V'. Since M^{12} is a dominating set, either (1) vertex v' is adjacent to some vertex $u' \in M^{12}$, or (2) $v' \in M^{12}$.

Case 1: v' is adjacent to some vertex $u' \in M^{12}$ (see Figure 4.9).

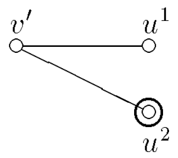


Figure 4.9: Case 1: Vertex v' must be adjacent to $u^1 \in M^1$. The circled vertex is in M^{12}.

Either $u' = u^1$ or $u' = u^2$. Therefore v' is adjacent to u^1 by Observation 4.22, and $u^1 \in M^1$ by Observation 4.24. It follows that v' is dominated by M^1.

Case 2.1: $v' \in M^{12}$ and $v^1 \notin M^{12}$ (see Figure 4.10).

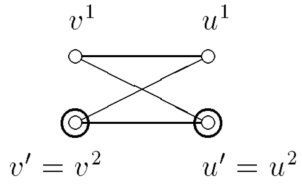


Figure 4.10: Case 2.1: Case 1 applies again. The circled vertices are in M^{12}.

Since v^1 is not in M^{12}, vertex v^1 must be adjacent to (dominated by) some vertex $u' \in M^{12}$. Therefore $v' = v^2$ is also adjacent to u' by Observation 4.22, and the previous case applies.

Case 2.2: $v' \in M^{12}$ and $v^1 \in M^{12}$ (see Figure 4.11).

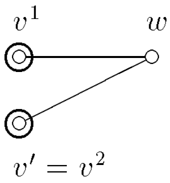


Figure 4.11: Case 2.2: Vertex v' is adjacent to a neighbour w^1 of v^1, both of which are in $f(M^{12})$. The circled vertices are in M^{12}.

If $v' = v^1$, then we are done. Otherwise, $v' = v^2$ and, by Equation 4.16, $f(v') = w^1$, where $w^1 \in V^1$ is some neighbour of v^1. Vertex v^2 is also adjacent to w^1 by Observation 4.22. Therefore v' is dominated by $M^1 = f(M^{12})$ since $w^1 = f(v')$ and $f(v') \in f(M^{12})$. ∎

Corollary 4.26.1 *If M^{12} is a minimum cardinality dominating set of a cocomparability graph G with no isolated vertices, then $M^1 = f(M^{12})$ is a minimum cardinality dominating set of G'.*

Proof: Immediate from the lemma and Observation 4.25. ∎

Lemma 4.27 *Let $G = (V, E)$ be a cocomparability graph with no isolated vertices. If $M^1 \subseteq V^1$ is a minimum cardinality dominating set of the t-auxiliary graph G', then $\sigma(M^1)$ is a minimum cardinality* total *dominating set of G.*

Proof: To see that $\sigma(M^1)$ is a total dominating set for G, let v be an arbitrary vertex in V. Then v^2 is adjacent to some vertex $u^1 \in M^1$, since M^1 dominates v^2 but M^1 does not contain any vertices from V^2. Therefore $v = \sigma(v^2)$ is adjacent to $\sigma(u^1)$ by Observation 4.20. Since $\sigma(u^1) \in \sigma(M^1)$, it follows that $\sigma(M^1)$ is a total dominating set of G.

To see that $\sigma(M^1)$ is of minimum cardinality, let T be a minimum cardinality total dominating set of G. By definition, every vertex in V is adjacent to some vertex in T. Let T^1 be the subset of V^1 that corresponds to T. That is, let $T^1 = \{t^1 : t \in T\}$. Clearly, every vertex in $v^1 \in V^1$ is adjacent to some vertex in T^1. Now consider an arbitrary vertex $v^2 \in V^2$. This vertex v^2 is adjacent to some vertex in T^1 since v^1 is adjacent to some vertex in T^1 and $\mathrm{Adj}(v^1, G') = \mathrm{Adj}(v^2, G')$ by Observation 4.22. Therefore T^1 is an (ordinary) dominating set of G', and $|\sigma(M^1)| = |M^1| \leq |T^1| = |T|$ by Corollary 4.26.1. ∎

Lemma 4.28 *Algorithm MCTDS-OCC (Table 4.2.3) prints a message stating that no total dominating set exists if and only if no total dominating set exists.*

Table 4.12: **Algorithm:** MCTDS-OCC(G) [Minimum Cardinality Total Dominating Set in spanning-Ordered Cocomparability Graphs]

Input: A spanning ordered cocomparability graph G.
Output: A minimum cardinality total dominating set M of G, or a message stating that no total dominating set exists.

1 **if** G has no isolated vertices,
2   **then** Construct the spanning-ordered t-auxiliary graph G'.
3         $M^{12} \leftarrow$ MCDS-OCC(G')
4         $M^1 \leftarrow \{f(v) : v \in M^{12}\}$ as defined by Equation 4.16.
5   **return** $\sigma(M^1)$.
6   **else** **return** "No total dominating set exists."

Proof: An isolated vertex of G cannot be total dominated, therefore no total dominating set exists in this case. On the other hand, $G(V)$ total dominates $G(V)$ if there are no isolated vertices, hence a total dominating set exists in this case. This condition is tested in Step 1. ∎

Theorem 4.29 *Algorithm MCTDS-OCC returns a minimum cardinality total dominating set of a spanning-ordered cocomparability graph if and only if one exists in $O(VE^2)$ time.*

Proof: The algorithm returns a set if and only if a minimum cardinality total dominating set exists by Lemma 4.28. The t-auxiliary graph G' constructed in Step 2 is a spanning-ordered graph by Lemma 4.23. Therefore the set M^{12} constructed in Step 3 is a minimum cardinality dominating set by Theorem 4.18. Since G does not have any isolated vertices, the set M^1 constructed in Step 4 is a minimum cardinality dominating set by Lemma 4.26. Finally, $\sigma(M^1)$ is a minimum cardinality total dominating set by Lemma 4.27.

To achieve the stated run time, represent all graphs with adjacency lists. Then Step 1 can be checked in $O(V)$ time. To construct the t-auxiliary graph in Step 2, begin by copying the graph in $O(V+E)$ time. Then construct E' by examining every edge $(u,v) \in E$ in $O(V+E)$ time and adding the edges (u^1,v^1), (u^2,v^2), and (u^1,v^2) in constant time. There are $2|E|$ edges in E^{12}, and $|E|$ edges in each of E^1 and E^2, for a total of $4|E|$ edges in E'. That is, G' has $O(E)$ edges, and since there are $2|V|$ vertices in V', graph G' has $O(V)$ vertices. Therefore, Step 3 can be executed in $O(VE^2)$ time by Theorem 4.18. Finally, the output in Steps 4 and 5 can be constructed in $O(V)$ time by examining each vertex in V and using the first available neighbour when one is needed. Step 3 is the time critical step and accounts for the complexity of the algorithm. ∎

4.2.4 Weighted Independent Dominating Sets

Until now, the problems in this section entailed finding minimum *cardinality* subgraphs. This section and the next (§4.2.5) generalizes this type of optimization problem by finding minimum *weight* subgraphs in graphs with weighted vertices. The weighted versions of the problems studied so far, namely minimum weight connected dominating set, minimum weight dominating set, and minimum weight total dominating set, all on cocomparability graphs, have been shown to be **NP**-complete by Maw Shang Chang [Cha94].

This section presents an efficient polynomial-time algorithm for finding a minimum weight independent dominating set in a weighted cocomparability graph. It does so by reducing the problem to finding a minimum weight maximal clique[1] in a comparability graph. It then exhibits a new efficient algorithm for finding a minimum weight maximal clique in a comparability graph.

Could we not negate all weights and find a *maximum* weight clique? Yes, certainly, but are standard maximum weight clique algorithms, for example the one in [Gol80], applicable here? They are not, because they are content to find a maximum weight clique, not necessarily a maximal one. A maximum weight clique is also a maximal

[1] Recall that a clique is a complete subgraph.

clique if its weights are positive. However, a clique found by such an algorithm would not contain any vertices with negative weights, even though such a clique might not be maximal. This is because the weight of any subgraph can be increased by removing any negative weight vertices.

On the other hand, the new algorithm presented in this section can also find a maximum weight maximal clique if the weights are first negated. Let us begin by exploring some relevant properties of independent dominating sets.

Observation 4.30 *An independent set is a dominating set if and only if it is a maximal independent set.*

Proof: Let I be an independent set. If I is a dominating set, then every vertex not in I is adjacent to some vertex in I. Therefore, I is a maximal independent set.

Conversely, if I is a maximal independent set, every vertex not in I must be adjacent to some vertex of I, and therefore is dominated by I. Since I trivially dominates all vertices in I, it is a dominating set. ∎

Observation 4.31 *A set is independent in a graph if and only if it is completely connected in the complement of the graph, and a set is maximal independent in a graph if and only if it is a maximal clique in the complement.*

The two observations above imply that we can find independent dominating sets by looking for maximal cliques in the complement. The complement of a cocomparability graph is, of course, a comparability graph. The next lemma shows what maximal cliques look like in comparability graphs. It requires the following definition from Section 1.3.1 (where it is phrased more generally in terms of relations).

Definition 4.32 A *transitive reduction* of a directed graph G is any directed graph, with the least number of edges, whose transitive closure is isomorphic to the transitive closure of G. ▮

Under this most general definition, a transitive reduction need not even be a subgraph of G. If, however, G is finite and acyclic, then the transitive reduction is a unique subgraph and may be obtained by systematically removing all transitively implied edges from G [AGU72]. Therefore the notion of *the* transitive reduction of a transitively oriented comparability graph is well defined.

Lemma 4.33 *The maximal cliques in a comparability graph are exactly the maximal paths in the transitive reduction of any transitive orientation of the graph.*

Proof: Let $G = (V, E)$ be a comparability graph, and let $T = (V, A)$ be any transitive orientation of G. Let the linear order $(V, <)$ be any linear extension of T.

Let C be any maximal clique of the comparability graph G. To see that C is a maximal (directed) path in the transitive reduction of T, let u and v be two vertices in C such that $u < v$. Then $(u, v) \in A$. Therefore the vertices of C, taken in their linear order, form a path in T. Furthermore, every arc (u, v) of this path must be in the transitive reduction of T. If it were not, there would be a path from u to v in T with at least one intermediate vertex, w. By the linear order, $u < w < v$, so that $w \notin C$. But by the transitivity of T, w would be adjacent in G to every vertex in C. That is, $C \cup \{w\}$ would be completely connected in G, contradicting the maximality of C. Finally, every path in T, and therefore in the transitive reduction also, is a completely connected subgraph of G by transitivity. Therefore C is a maximal path in T since any superpath of C would contradict the maximality of C in G.

Conversely, let $P = (p_1, \ldots, p_k)$ be a maximal path in the transitive reduction of T. Then P is a clique in G by transitivity, and $p_1 < p_2 < \cdots < p_k$. Suppose that P is not a *maximal* clique in G. Then there exists a vertex $v \in V \setminus P$ that is adjacent in G to every vertex in P. If $v < p_1$, then there is an arc from v to p_1 in T. Therefore there is a path from v to p_1 in the transitive reduction of T. This contradicts the assumption that P is a maximal path. If $v > p_k$, then a similar contradiction arises. Finally, if $p_1 < v < p_k$, then there are two vertices u and w in P such that $u < v < w$. Therefore there are arcs from u to v, and from v to w, in T. But this contradicts the assumption that (u, w) is an arc in the transitive reduction of T. ∎

These lemmas have set the stage for the polynomial time algorithm below. They show that independent dominating sets in cocomparability graphs are precisely the paths from minimal elements to maximal elements in the transitive reduction of a transitive orientation of the complement of the graph.

To simplify the handling of special cases[2], augment the transitively oriented graph with two sentinel vertices, $s = 0$ and $t = |V| + 1$, that have 0 weight. Add an arc from s to every other vertex (including t), and add an arc to t from every other vertex (including s). Therefore s is the only minimal element and t is the only maximal element. Note that s and t must appear in every maximal clique C and that $C \setminus \{s, t\}$ is a maximal clique in the unaugmented graph. We want only the minimum weight maximal clique, that is, the minimum weight path from s to t. The details of the algorithm appear in Table 4.2.4.

Theorem 4.34 *Algorithm MWMC-C finds a minimum weight maximal clique in a weighted comparability graph $G = (V, E, w)$ in $O(M(V))$ time.*

Proof: Correctness follows from Lemma 4.33 and the discussion concerning augmenting the digraph.

Step 1 runs in $O(V^2)$ time [Spi85], and Step 2 runs in $O(V)$ time. Since the augmented

[2] We can achieve the same effect by augmenting the transitive reduction R with s and t. In this case we would add arcs only from s to minimal elements of R and from maximal elements of R to t. Instead, we have chosen to let the transitive reduction step find the minimal and maximal elements for us.

Table 4.13: **Algorithm:** MWMC-C(G) [Minimum Weight Maximal Clique for Comparability Graphs]

Input: A weighted comparability graph G.
Output: A minimum weight maximal clique in G.

1 $T \leftarrow$ a transitive orientation of G.
2 augment T with $s = 0$ and $t = |V| + 1$.
3 $R \leftarrow$ the transitive reduction of T.
4 $C \leftarrow$ the least weight path from s to t in R.
5 **return** $C \setminus \{s, t\}$.

T is already transitively closed, the transitively implied edges are given by T^2. We can find this by creating the adjacency matrix for T and multiplying this by itself in $O(M(V))$ time. Then $R = T \setminus T^2$ can be computed in $O(V^2)$ time, and so Step 3 runs in $O(M(V))$ time[3]. Let the weight of an arc $(u, v) \in R$ be $w((u, v)) = w(u)$. Then the edge weight of a path from s to t is equal to its vertex weight since $w(s) = w(t) = 0$. Therefore Step 4 can be implemented to run in $O(V + E)$ time since R is a directed acyclic graph [CLR90]. The entire algorithm therefore runs in $O(M(V))$ time. ∎

We are now ready to construct algorithm MWIDS-CC for finding an optimal independent dominating set in a cocomparability graph, as shown in Table 4.14.

Theorem 4.35 *Algorithm MWIDS-CC finds a minimum weight independent dominating set in a weighted cocomparability graph $G = (V, E, w)$ in $O(M(V))$ time.*

Proof: Correctness follows from Observation 4.30, Observation 4.31, and Theorem 4.34. Step 1 runs in $O(V^2)$ time, and Step 2 runs in $O(M(V))$ time (Theorem 4.34). ∎

[3]The transitive reduction of an acyclic digraph $D = (V, A)$ can actually be computed in $O(kR) = O(VR) = O(VA)$ time [Sim88], where k is the number of paths in a path decomposition of D. Since R is not reflected in the output of Algorithm MWMC-C, this proof uses the $O(M(V)) = O(V^{2.376})$ bound.

Table 4.14: **Algorithm:** MWIDS-CC(G) [Minimum Weight Independent Dominating Set for Cocomparability Graphs]

Input: A weighted cocomparability graph G.
Output: A minimum weight independent dominating set of G.

$\overline{G} \leftarrow$ the complement of G.
$I \leftarrow$ MWMC-C($\overline{G}$)
return I.

4.2.5 Weighted Steiner Sets

This section describes a polynomial time algorithm for finding a minimum weight Steiner set (MWSS) in a cocomparability graph. This algorithm can be implemented to run in $O(V \log V + E)$ time on a vertex-weighted spanning-ordered cocomparability graph $G = (V, E, <, w)$. The algorithm terminates in $O(V^2)$ time even if a spanning order is not available. Furthermore, given an arbitrary graph, the algorithm will either find a minimum weight Steiner set, or it will be print a message stating that the graph is not a cocomparability graph. That is, the algorithm does not need to recognize cocomparability graphs, a step that would add $\Theta(M(V))$ to the run-time complexity.

These results simplify and generalize the permutation graph algorithms obtained by Colbourn and Stewart [CS90a] ($O(V^3)$ time) and by Arvind and Rangan [AR92] ($O(V \log V + E)$ time). They also extend an $O(V + E)$ algorithm for minimum *cardinality* Steiner sets in cocomparability graphs due to Colbourn and Lubiw (described in [KS93]).

Definition 4.36 Let $G = (V, E, w)$ be a graph with nonnegatively weighted vertices. Let $R \subseteq V$ be a "required" set of vertices in G. A *Steiner set* (of G and R) is a set S of vertices such that (1) $S \subseteq V \setminus R$ and (2) the subgraph $G(S \cup R)$ induced in G by $S \cup R$

is connected. The *weight* $w(S)$ of a set of vertices S is defined, as usual, to be the sum of the weights of its elements. Assume without loss of generality that $w(v) = 0$ for all vertices $v \in R$. The *weighted Steiner set problem* is to find a minimum weight Steiner set in such a vertex weighted graph. Finally, if G is linearly ordered, let $s = \min R$ be the least point in R and $t = \max R$ be the greatest point in R. ∎

Lemma 4.37 *Let P be a path from s to t in a spanning-ordered cocomparability graph. Then $S = P \setminus R$ is a Steiner set.*

Proof: Since $S \subseteq V \setminus R$, we only need to show that $S + R$ is connected. First note that $S + R = P \cup R$. Since P is connected, it suffices to show that every vertex $v \in R \setminus P$ is adjacent to some vertex in P. This follows from Lemma 4.7. ∎

Lemma 4.38 *Let G be an arbitrary weighted graph $G = (V, E, w)$ and $R \subseteq V$ be a required set. Let P be a least weight path from some vertex $u \in R$ to some vertex $v \in R$. If S is a Steiner set of G and R, and $S \subset P$, then S is a minimum weight Steiner set.*

Proof: Let S' be a minimum weight Steiner set of G and R. Since the subgraph induced by $S' \cup R$ is connected, there is a path P' from u to v in this subgraph, and $w(P') \leq w(S' \cup R)$. Since P is a least weight path, $w(P) \leq w(P')$. Now $w(S' \cup R) = w(S')$ since $w(v) = 0$ if $v \in R$. Finally, $w(S) \leq w(P)$ since $S \subset P$. It follows that $w(S) \leq w(P) \leq w(P') \leq w(S' \cup R) = w(S')$. That is, S is also a minimum weight Steiner set. ∎

Theorem 4.39 *Let P be a least weight path from s to t in a spanning-ordered cocomparability graph G. Then $S = P \setminus R$ is a minimum weight Steiner set.*

Proof: The set S is a Steiner set by Lemma 4.37 and therefore a minimum weight Steiner set by Lemma 4.38. ∎

One way to use this result for an arbitrary graph is to first check if G is a cocomparability graph, then construct a spanning order, and find a least weight path. The best known upper bound for recognizing an order n cocomparability graph is the same as that for $n \times n$ matrix multiplication [Spi85]. Since this bound is currently $O(n^{2.376})$ [CW87], recognition is the limiting step in this approach. Spinrad [Spi85] shows that the complexity of recognizing comparability graphs and transitive digraphs is the same.[4] Therefore, we are no better off by assuming a linearly ordered graph as input, since recognizing a spanning-ordered graph would entail recognizing the transitively oriented complement. Fortunately, there is another approach, which does not require recognition.

Table 4.15: **Algorithm:** MWSS-OCC(G, R) [Minimum Weight Steiner Set for spanning-Ordered Cocomparability Graphs]

Input: A linearly-ordered vertex-weighted graph $G = (V, E, <, w)$ and a required set of vertices $R \subseteq V$.
Output: A minimum weight Steiner set S,
or a message stating that no Steiner set exists,
or a message stating that G is not a spanning-ordered cocomparability graph.

if R is empty,
  then return the empty set
    halt.
if R is not a subset of a connected component of G,
  then return "No Steiner set exists."
    halt.
$P \leftarrow$ a least weight path from $\min R$ to $\max R$ in G.
$S \leftarrow P \setminus R$
if the subgraph induced by $S + R$ is connected,
  then return S
  else return "G is not a spanning-ordered cocomparability graph."

Lemma 4.40 *Given a linearly-ordered graph G and required vertices R, Algorithm*

[4] More precisely, if transitive digraphs can be recognized in $O(f(V))$ time, then comparability graphs (and therefore also cocomparability graphs) can be recognized in $O(f(V) + V^2)$ time. Conversely, if comparability graphs can be recognized in $O(f(n))$ time, then transitive digraphs can be recognized in $O(f(n) + E)$ time.

MWSS-OCC (Table 4.2.5) prints a message stating that no Steiner set exists if and only if no Steiner set exists.

Proof: A Steiner set exists if and only if the required vertices R can be augmented to induce a connected subgraph. This is true if and only if the vertices of R all lie in the same connected component of G. This condition is checked in Step 4. ∎

Lemma 4.41 *Given a linearly-ordered graph G and required vertices R, if Algorithm MWSS-OCC returns a set S, then S is a minimum weight Steiner set.*

Proof: Suppose the algorithm returns a set S. This can only happen in Steps 2 and 10. If S is returned by Step 2, then R is empty. A minimum weight Steiner set can therefore also be empty since all vertex weights are nonnegative.

If S is returned by Step 10, then the subgraph induced by $S + R$ is connected. So S is a Steiner set by definition, and therefore a minimum weight Steiner set by Lemma 4.38. ∎

Theorem 4.42 *Given a spanning-ordered cocomparability graph G and required set $R \subseteq V$, Algorithm MWSS-OCC returns a minimum weight Steiner set of G and R if and only if it exists.*

Proof: If a minimum weight Steiner set does not exist, then no Steiner set exists since the graph is finite. By Lemma 4.40, a message is printed and the algorithm halts at Step 3.

Suppose, on the other hand, that such a set does exist. If R is empty, a suitable minimum weight Steiner set, the empty set, is returned by Step 2. Otherwise, the set $S = P \setminus R$ constructed in Step 8 is a minimum weight Steiner set by Theorem 4.39. That is, the subgraph induced by $S + R$ is connected. So S is correctly returned by Step 10. ∎

Theorem 4.43 *Given any linearly-ordered graph $G = (V, E, <, w)$, Algorithm MWSS-OCC runs in $O(V \log V + E)$ time.*

Proof: Use linked lists to represent all sets, and represent the graph as an adjacency list. Then Step 1 can be executed in constant time.

Step 4 can be implemented by marking all vertices in the same connected component as the first vertex in R using a single phase of depth first search in $O(E)$ time. It then suffices to check, in $O(R)$ time, that all vertices in R are marked. Step 4 therefore executes in $O(V + E)$ time.

The extreme vertices $\min R$ and $\max R$ can easily be found in $O(R) = O(V)$ time. Step 7 uses Dijkstra's algorithm with a Fibonacci heap priority queue [CLR90] to find a least weight path between $\min R$ and $\max R$ in $O(V \log V + E)$ time. Dijkstra's algorithm expects an edge-weighted digraph as input. So, implicitly set the weight of a directed edge to be $w((u, v)) = w(u)$. Since $w(\min R) = w(\max R) = 0$, the edge weight of a simple path from $\min R$ to $\max R$ is equal to its vertex weight.

Step 8, setting $S = P \setminus R$, can be computed in $O(P + R) = O(V)$ time. For example, the algorithm could mark all vertices in R and let S be the set of unmarked vertices in P.

Finally, Step 9 represents the induced subgraph in $O(S + R) = O(V)$ time by marking the inducing vertices in G. It then tests the induced subgraph for connectivity in $O(V + E)$ time by depth first searching the marked vertices. ∎

Although Algorithm MWSS-OCC may work—find minimum weight Steiner sets—for some arbitrary linearly-ordered graphs and sets of required vertices, the class of graphs for which it works for all sets of required vertices is exactly the spanning-ordered cocomparability graphs. This is restated by the following lemma.

Lemma 4.44 *For every linearly-ordered graph that is not a spanning-ordered cocomparability graph, there exists a required set such that Algorithm MWSS-OCC prints the message stating that the graph is not a spanning-ordered cocomparability graph.*

Proof: Let $G = (V, E, <, w)$ be a linearly-ordered graph that is not a spanning-ordered graph. Then there exists three vertices $a < b < c$ that violate the spanning order, that is, $(a, c) \in E$ but $(a, b) \notin E$ and $(b, c) \notin E$. Let $R = \{a, b, c\}$. Algorithm MWSS-OCC will find the least weight path $P = (\min R, \max R) = (a, b)$ in Step 7, and it will construct the empty set S in Step 8. But the subgraph induced by $S + R = R$ is not connected since b is not adjacent to either a or c. Hence the message will be printed in Step 11. ∎

We now have enough tools to find a minimum weight Steiner set in a cocomparability graph. Again, the input to the algorithm will be an arbitrary graph; the algorithm does not require a recognition step.

Table 4.16: **Algorithm:** MWSS-CC(G, R) [Minimum Weight Steiner Set for Cocomparability Graphs]

Input: A weighted graph $G = (V, E, w)$ and a required set of vertices $R \subseteq V$.
Output: A minimum weight Steiner set S,
or a message stating that no Steiner set exists,
or a message stating that G is not a cocomparability graph.

```
1   G ← OCC(G)
2   return MWSS-OCC(G, R)
```

Theorem 4.45 *Given a weighted graph G and a required set of vertices R, Algorithm MWSS-CC (Table 4.2.5) computes a minimum weight Steiner set, or prints a message stating that no Steiner set exists, or prints a message stating that G is not a cocomparability graph, in $O(V^2)$ time.*

Proof: By Theorem 4.5, Step 1 either returns a linearly-ordered graph, or correctly prints a message stating that G is not a cocomparability graph. If Step 2 returns a set S, then S is a minimum weight Steiner set by Lemma 4.41. If it prints a message stating that no Steiner set exists, then no Steiner set exists by Lemma 4.40. Finally, if it prints a message stating that G is not a spanning-ordered cocomparability graph, then G is also not a cocomparability graph for, if it were, it would have been spanning-ordered by Step 1.

Step 1 takes $O(V^2)$ time by Theorem 4.5, and Step 2 takes $O(V \log V + E)$ time by Theorem 4.43. ∎

4.2.6 Applications

We can use the algorithms for spanning-ordered graphs to solve the corresponding problems for permutation graphs, interval graphs, and indifference graphs, given permutation, interval, or indifference realizations of the graphs. These graphs are all cocomparability graphs, and as shown here, a spanning order can easily be extracted from their realizations.

Theorem 4.46 *All spanning-ordered cocomparability graph algorithms work without change for permutation graphs given a permutation realization linearly ordered by vertex label.*

Proof: Let $G = (V, E)$ be a permutation graph, and let the labelling of the vertices and π be a permutation realization. That is, π is a permutation of $\{1, 2, \ldots, n\}$ such that $(u, v) \in E$ if and only if $(u - v)(\pi^{-1}(u) - \pi^{-1}(v)) < 0$. Then the vertex labelling is a spanning order.

To see this, let $u < v < w$ be three vertices such that u and w are adjacent in G. It follows that $(u - w)(\pi^{-1}(u) - \pi^{-1}(w)) < 0$, which implies that $\pi^{-1}(u) > \pi^{-1}(w)$ since

$u < w$. Either $\pi^{-1}(v) < \pi^{-1}(u)$ or $\pi^{-1}(v) > \pi^{-1}(u)$. If $\pi^{-1}(v) < \pi^{-1}(u)$, then $(u, v) \in E$ since $u < v$. Otherwise, $\pi^{-1}(v) > \pi^{-1}(u)$ and $\pi^{-1}(u) > \pi^{-1}(w)$ since $(u, w) \in E$. Therefore $\pi^{-1}(v) > \pi^{-1}(w)$ so that $(v, w) \in E$. That is, u and v are adjacent, or v and w are adjacent. ▮

Theorem 4.47 *All spanning-ordered cocomparability graph algorithms work without change for interval graphs given a set of intervals in nondecreasing order of left endpoint.*

Proof: Such a linear ordering of intervals is a spanning order. Let $u \le v \le w$ be three intervals such that u and w are adjacent. Then the left endpoint of w must be to the left of the right endpoint of u. Therefore the left endpoint of v lies between the left endpoint of u and the right endpoint of u. That is, u and v are adjacent. ▮

Theorem 4.48 *All spanning-ordered cocomparability graph algorithms work without change for τ-strip graphs, given a set of points in order of nondecreasing x-coordinate, for all $\tau \in [0, \sqrt{3}/2]$.*

Proof: Such a linear ordering of points is a spanning order. Let $f : V \to \{0, \tau\}$ be a strip realization. Let $u \le v \le w$ be three points such that u and w are adjacent. Then by the definition of τ-strip graphs, $x_f(w) - x_f(u) \le 1$. This implies that either $x_f(v) - x_f(u) \le 1/2$ or $x_f(w) - x_f(v) \le 1/2$. If the former, then

$$\begin{aligned}
\|f(v) - f(u)\|^2 &= (x_f(v) - x_f(u))^2 + (y_f(v) - y_f(u))^2 \\
&\le 1/4 + \tau^2 \\
&\le 1/4 + (\sqrt{3}/2)^2 \\
&= 1.
\end{aligned}$$

That is, u and v are adjacent. On the other hand, if $x_f(w) - x_f(v) \le 1/2$, then v and w are adjacent. ▮

Corollary 4.48.1 *All spanning-ordered cocomparability graph algorithms work without change for indifference graphs given a set of points in nondecreasing order.*

Proof: Indifference graphs are 0-strip graphs. ∎

4.3 Transitive Orientation, Implication Classes, and Dominating Paths

This section explores the possible transitive orientations for the nonedges of cocomparability graphs. Chapter 5 and Chapter 6 use the transitive orientations of the complements of τ-strip graphs and two-level graphs in characterizing these classes of graphs. For example, Chapter 6 uses the results of this section to show that two-level graphs have essentially one such orientation that is compatible with a given assignment of y-coordinates for the vertices.

Subsection 4.3.1 establishes some basic properties of transitive orientations in cocomparability graphs. In particular, it shows how the orientations of some nonedges "force" the orientation of others. With this background, Subsection 4.3.2 then studies cocomparability graphs that have induced dominating paths with four or more vertices.

4.3.1 Definitions and Basic Properties

The following notation follows that of Golumbic ([Gol80] pages 105–148), where it is used for comparability graphs. Since τ-strip graphs are *cocomparability* graphs, this section modifies Golumbic's notation for the complement of comparability graphs. Let $G = (V, E)$ be an undirected graph. Define the *complementary arcs (nonedges)* $\overline{E}$ to be the set of distinct *ordered* pairs not in E, that is,

$$\overline{E} = \{(u, v) : u \neq v \text{ and } (u, v) \notin E\}.$$

Definition 4.49 Write $(a, b)\Gamma(c, d)$, and say that arc (a, b) *directly forces* arc (c, d), if

1. $a = c$ and $(b, d) \notin \overline{E}$, or
2. $b = d$ and $(a, c) \notin \overline{E}$.

That is, Γ is a binary relation on $\overline{E}$. Say that (a, b) *forces* (c, d), and write $(a, b)\Gamma^*(c, d)$, if there exists a sequence of arcs (a *forcing chain of length* $k \geq 0$) such that

$$(a, b) = (a_0, b_0)\Gamma(a_1, b_1)\Gamma \cdots \Gamma(a_k, b_k) = (c, d).$$

That is, Γ^* is the transitive closure of Γ. ∎

Note that Γ is reflexive and symmetric. Note also that $(a, b)\Gamma(c, d)$ if and only if $(b, a)\Gamma(d, c)$. The following lemma explains the intent of these definitions.

Lemma 4.50 *Let $G = (V, E)$ be a cocomparability graph and $\vec{G} = (V, \vec{E})$ be a transitive orientation of G. If (a, b) forces (c, d), then $(a, b) \in \vec{E}$ if and only if $(c, d) \in \vec{E}$.*

Proof: The lemma follows by induction on the length k of the forcing chain. The base case $k = 0$ is trivial. For the inductive case, consider the forcing chain

$$(a, b) = (a_0, b_0)\Gamma(a_1, b_1)\Gamma \cdots \Gamma(a_{k-1}, b_{k-1})\Gamma(a_k, b_k) = (c, d),$$

and assume $(a_0, b_0) \in \vec{E}$ if and only if $(a_{k-1}, b_{k-1}) \in \vec{E}$. Since $(a_{k-1}, b_{k-1})\Gamma(a_k, b_k)$, either $a_{k-1} = a_k$ and $(b_{k-1}, b_k) \notin \overline{E}$, or $b_{k-1} = b_k$ and $(a_{k-1}, a_k) \notin \overline{E}$. If $a_{k-1} = a_k$ and $(b_{k-1}, b_k) \notin \overline{E}$, then $(a_{k-1}, b_{k-1}) \in \vec{E}$ if and only if $(a_k, b_k) \in \vec{E}$. For otherwise, transitivity of $\vec{E}$ implies $(b_{k-1}, b_k) \in \vec{E}$ or $(b_k, b_{k-1}) \in \vec{E}$, and therefore $(b_{k-1}, b_k) \in \overline{E}$; see Figure 4.12. Similarly, if $b_{k-1} = b_k$ and $(a_{k-1}, a_k) \notin \overline{E}$, then $(a_{k-1}, b_{k-1}) \in \vec{E}$ if and only if $(a_k, b_k) \in \vec{E}$. It follows that $(a_0, b_0) \in \vec{E}$ if and only if $(a_k, b_k) \in \vec{E}$. ∎

Note that Γ^* is an equivalence relation, i.e., reflexive, symmetric, and transitive. It therefore partitions $\overline{E}$ into equivalence classes that we refer to as *implication classes*. Let $[e]$ denote the implication class generated by $e \in \overline{E}$. The following theorems will be useful later, when orienting two-level graphs in Chapter 6.

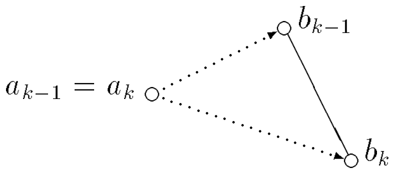


Figure 4.12: $(a_{k-1}, b_{k-1}) \in \vec{E}$ if and only if $(a_k, b_k) \in \vec{E}$.

Theorem 4.51 ([Gol80], page 107) *Let A be an implication class of [the complementary arcs of] an undirected graph G. If G has a transitive orientation F, then either $F \cap (A \cup A^{-1}) = A$ or $F \cap (A \cup A^{-1}) = A^{-1}$ and, in either case, $A \cap A^{-1} = \emptyset$.*

Theorem 4.52 ([Gol80], page 122) *Let $G = (V, E)$ be a graph. Then G is a cocomparability graph if and only if $[e] \cap [e]^{-1} = \emptyset$ for all $e \in \overline{E}$.*

To illustrate Theorem 4.52 in action, let us reprove Theorem 4.6 (from the introduction to this chapter). An examination of both proofs will also associate the notion of forcing chains with the notion of spanning order.

Theorem 4.6 (§4.1) *Cocomparability graphs do not have induced cycles with five or more edges.*

Proof: Let $(v_1, v_2, \ldots, v_n, v_1)$ be an induced cycle in a graph G, where $n \geq 5$, as shown in Figure 4.13. Then $(v_1, v_3) \in \overline{E}$ and $(v_3, v_1) \in [(v_1, v_3)]$, as demonstrated by the forcing chain

$$(v_1, v_3)\Gamma(v_1, v_4)\Gamma \ldots \Gamma(v_1, v_{n-1})\Gamma(v_2, v_{n-1})\Gamma(v_2, v_{n-1})\Gamma(v_2, v_n)\Gamma(v_3, v_n)\Gamma(v_3, v_1).$$

Therefore $[(v_1, v_3)] \cap [(v_1, v_3)]^{-1}$ is not empty, and G is not a cocomparability graph by Theorem 4.52. ∎

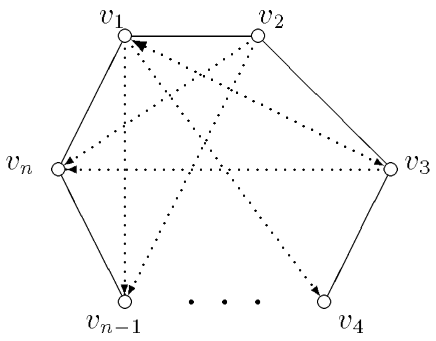


Figure 4.13: An induced cycle on $n \geq 5$ vertices. The arrows show that $(v_1, v_3)\Gamma^*(v_3, v_1)$.

4.3.2 Transitive Orientation and Dominating Paths

When there is a long dominating path in a graph, every nonedge either has its orientation forced by the nonedge joining the path's endpoints, or belongs to a nearly-oriented square or claw. Recall that a path is *chordless* if it is simple and if its vertices are adjacent in the graph only if they are adjacent in the path. For example, consider the connected τ-strip graph ($0 \leq \tau \leq \sqrt{3}/2$) generated by some set of points in a strip. A shortest path from the leftmost point to the rightmost point in the strip is chordless, otherwise there would be a shorter path. Furthermore, if the leftmost and rightmost points are more than three Euclidean units apart, then the path has at least three edges (and at least four vertices).

The complementary arc between the endpoints of a chordless path force all other complementary arcs with endpoints on the path, as shown by the following lemma. This easy lemma will be so useful to us that we will often use it without reference.

Lemma 4.53 (The Path Lemma) *If* $(s = p_0, p_1, \ldots, p_k = t)$ *is a chordless path in an arbitrary graph, then* (s, t) *forces* (p_i, p_j) *whenever* $i < j$.

Proof: Since the p_i are distinct and adjacent only to their neighbours in the path, there is a sequence of arcs

$$(s,t) = (s,p_k)\Gamma(s,p_{k-1})\Gamma\cdots\Gamma(s,p_{j+1})\Gamma(s,p_j),$$

so that (s,t) forces (s,p_j), as shown in Figure 4.14. There is also a sequence of arcs

$$(s,p_j) = (p_0,p_j)\Gamma(p_1,p_j)\Gamma\cdots\Gamma(p_{i-1},p_j)\Gamma(p_i,p_j),$$

so that (s,p_j) forces (p_i,p_j). The lemma follows since Γ^* is transitive. ▮

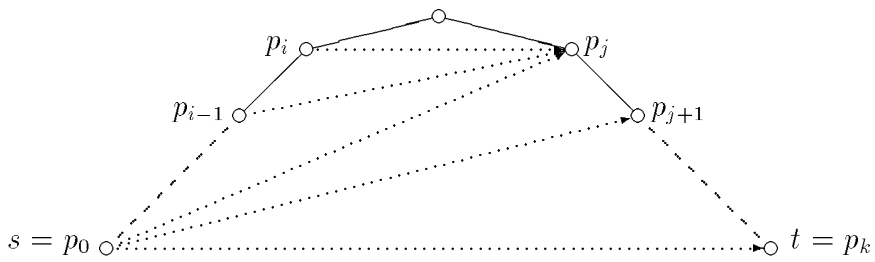


Figure 4.14: The endpoints of a chordless path force the chords in the complement.

Definition 4.54 A *colour class* of a graph G is the symmetric closure of an implication class. That is, if $[a]$ is an implication class of $G=(V,E)$, where $a \in \overline{E}$, then

$$\widehat{[a]} = [a] \cup [a]^{-1}$$

is a colour class. ▮

The following theorem relates colour classes to cocomparability graphs.

Theorem 4.55 ([Gol80], page 109) *Each [undirected subgraph induced by a] colour class of a graph G either has exactly two transitive orientations, one being the reversal of the other, or has no transitive orientation. If in G there is a colour class having no transitive orientation, then G fails to be a cocomparability graph.*

It is sometimes the case that a graph has only one colour class. If such a graph is a cocomparability graph, then it is said to be *uniquely complement orientable*[5]. This section deals with graphs that are not so strongly constrained. We will see (Theorem 4.57) that if a graph contains a sufficiently long chordless dominating path, then every complementary arc is either forced or captured—trapped in a small, otherwise forced subgraph. That is, the graph has an implication class such that every other arc is captured by it. The following definition is more precise.

Definition 4.56 Let $G = (V, E)$ be a graph, and let (s, t) be an arc in its complement $\overline{G} = (V, \overline{E})$. Say that (s, t) *captures* (u, v) if $(u, v) \in \overline{E}$ is part of one of the three induced subgraphs in Figure 4.15, where the directed nonedges are forced by (s, t). ▮

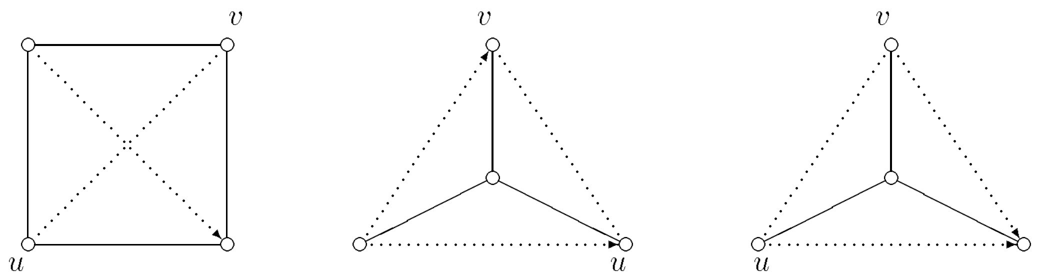


Figure 4.15: (s, t) captures (u, v).

Theorem 4.57 (The Capture Theorem) *Let $G = (V, E)$ be a connected graph, and let $P = (s, \ldots, t)$ be a chordless dominating path in G. If P has at least four vertices, then for all $(u, v) \in \overline{E}$, either (s, t) forces (u, v); (s, t) forces (v, u); or (s, t) captures (u, v).*

[5]Golumbic [Gol80] would refer to the complement, which is a comparability graph, as *uniquely partially orderable*. I have chosen not to use this term since it is easily confused with the spanning order, which is a linear order.

Proof: Linearly order the vertices in P according to their distance from s (so that s is the least vertex and t the greatest). Let u' be the least vertex in P that dominates u, as shown in Figure 4.16. It may be that $u' = u$, but in any event $(u, u') \notin \overline{E}$.

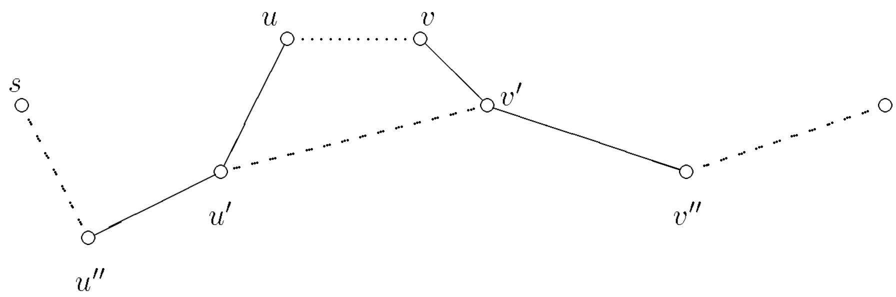


Figure 4.16: Definition of u' and v'.

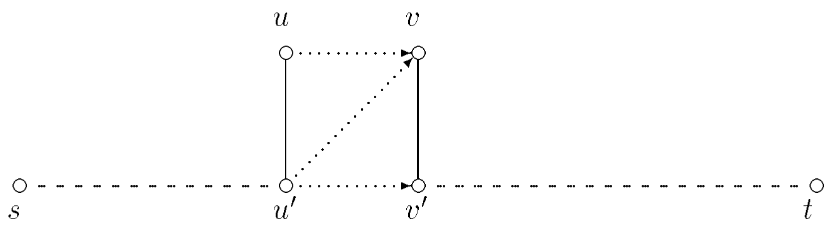
Assume without loss of generality that v is not dominated by any vertex in P less than u' (otherwise swap labels u and v and recompute u'). Let v' be the greatest vertex in P that dominates v. Note that v' is not less than u'. Again, it may be that $v' = v$, but in any event $(v, v') \notin \overline{E}$. However, $u \neq v'$, since $(u, v) \in \overline{E}$ but $(u, u') \notin \overline{E}$. Similarly, $v \neq u'$. The definitions and assumptions above also pertain to Lemmas 4.58 through 4.61.

There are three alternatives: $(u', v') \in \overline{E}$, $(u', v') \in E$, or $u' = v'$. The following three lemmas prove the theorem: if $(u', v') \in \overline{E}$, then Lemma 4.58 applies; if $(u', v') \in E$, then Lemma 4.60 applies; and if $u' = v'$, then Lemma 4.61 applies. ∎

Lemma 4.58 *Let $G = (V, E)$ be a connected graph, and let $P = (s, \ldots, t)$ be a chordless dominating path in G. If P has at least four vertices, then for all $(u, v) \in \overline{E}$* for which $(u', v') \in \overline{E}$*, either (s, t) forces (u, v), or (s, t) captures (u, v).*

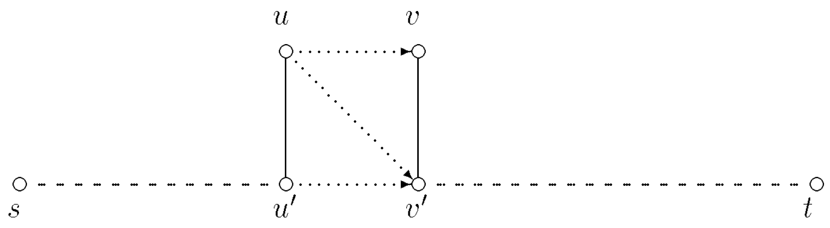
Proof: Case 1: $(u', v) \in \overline{E}$.

Then $(s, t)\Gamma^*(u', v')\Gamma(u', v)\Gamma(u, v)$, so that (s, t) forces (u, v).



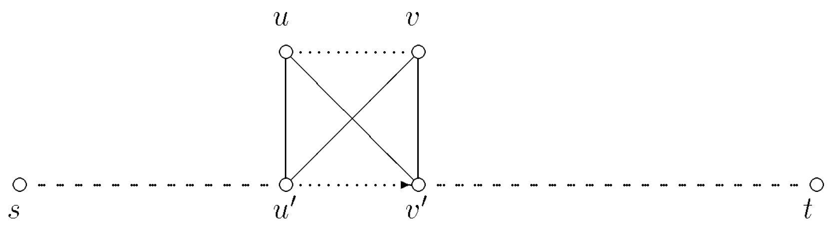
Case 2: $(u, v') \in \overline{E}$.

Then $(s, t)\Gamma^*(u', v')\Gamma(u, v')\Gamma(u, v)$, so again, (s, t) forces (u, v).



Case 3: $(u', v) \notin \overline{E}$ and $(u, v') \notin \overline{E}$.

Then since $(u, v) \in \overline{E}$, $u' \neq u$ and $v' \neq v$. Therefore $\{u, v, u', v'\}$ induces a square in G, and (s, t) captures (u, v).



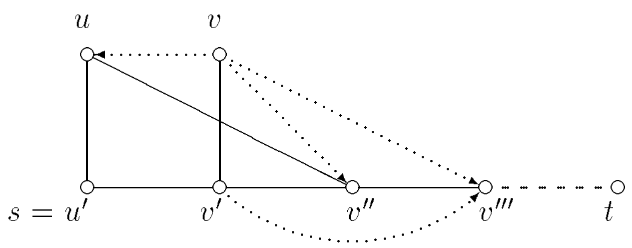
∎

Lemma 4.59 *Let $G = (V, E)$ be a connected graph, and let $P = (s, \ldots, t)$ be a chordless dominating path in G. If P has at least four vertices, then for all $(u, v) \in \overline{E}$* for which $(u', v') \in E$ and $u' = s$, *either (s, t) forces (u, v); (s, t) forces (v, u); or (s, t) captures (u, v).*

Proof: Since $u' = s$ and P has at least four vertices, $P = (s = u', v', v'', v''', \ldots, t)$. Let us proceed by case.

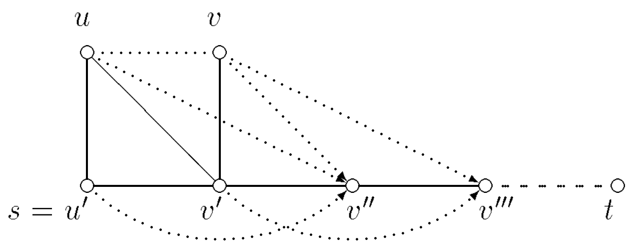
Case 1: $(u, v'') \notin \overline{E}$.

In this case, (s, t) forces (v, u) since $(s, t)\Gamma^*(v', v''')\Gamma(v, v''')\Gamma(v, v'')\Gamma(v, u)$.



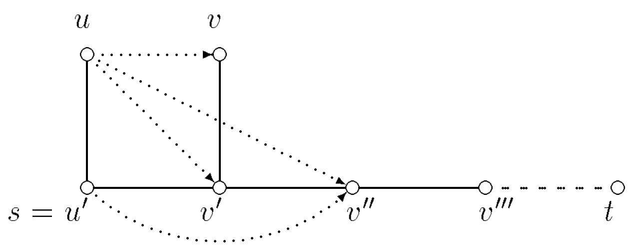
Case 2: $(u, v'') \in \overline{E}$ and $(u, v') \notin \overline{E}$.

Then $\{v', u, v, v''\}$ induces a claw in G. Furthermore, $(s, t)\Gamma^*(u', v'')\Gamma(u, v'')$ and $(s, t)\Gamma^*(v', v''')\Gamma(v, v''')\Gamma(v, v'')$, so (s, t) captures (u, v).



Case 3: $(u, v'') \in \overline{E}$ and $(u, v') \in \overline{E}$.

Here, (s, t) forces (u, v) since $(s, t)\Gamma^*(u', v'')\Gamma(u, v'')\Gamma(u, v')\Gamma(u, v)$.



∎

Corollary 4.59.1 *Let $G = (V, E)$ be a connected graph, and let $P = (s, \ldots, t)$ be a maximum dominating path in G. If P has at least four vertices, then for all $(u, v) \in \overline{E}$* for which $(u', v') \in E$ and $v' = t$, *either (s, t) forces (u, v); (s, t) forces (v, u); or (s, t) captures (u, v).*

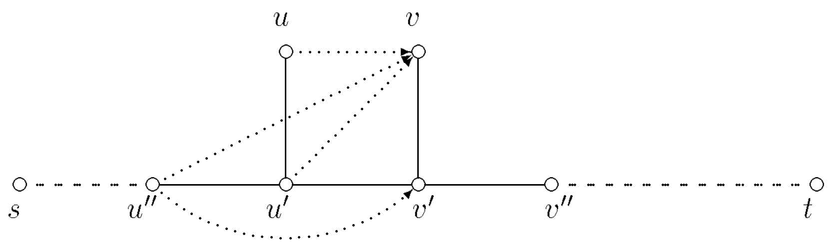
Proof: The lemma applies if we reverse path P, swap labels s and t, and swap labels u and v. ∎

Lemma 4.60 *Let $G = (V, E)$ be a connected graph, and let $P = (s, \ldots, t)$ be a chordless dominating path in G. If P has at least four vertices, then for all $(u, v) \in \overline{E}$* for which *$(u', v') \in E$, either (s, t) forces (u, v); (s, t) forces (v, u); or (s, t) captures (u, v).*

Proof: If $u' = s$ or $v' = t$, then Lemma 4.59 or its corollary applies, and we are done. So assume $P = (s, \ldots, u'', u', v', v'', \ldots, t)$. Note that $(u'', v) \in \overline{E}$ since v is not dominated by any vertex less than u'.

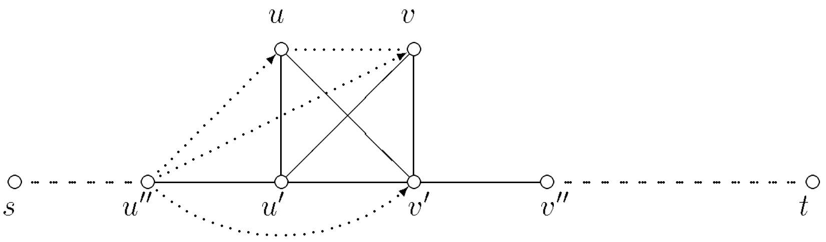
Case 1: $(u', v) \in \overline{E}$.

Here (s, t) forces (u, v) since $(s, t)\Gamma^*(u'', v')\Gamma(u'', v)\Gamma(u', v)\Gamma(u, v)$.



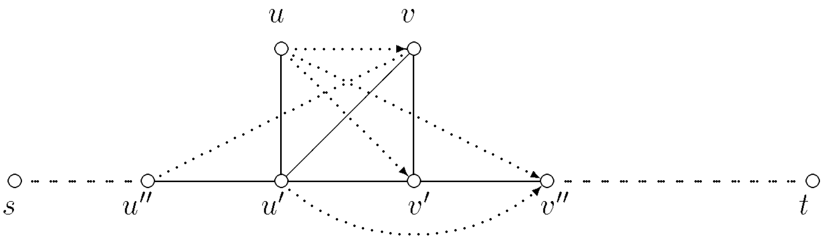
Case 2: $(u', v) \notin \overline{E}$ and $(u, v') \notin \overline{E}$.

It cannot be that $u' = u$ since $(u, v) \in \overline{E}$, but $(u', v) \notin \overline{E}$. Therefore $\{u', u'', u, v\}$ induces a claw in G. Furthermore, $(s,t)\Gamma^*(u'', v')\Gamma(u'', v)$, and $(s,t)\Gamma^*(u'', v')\Gamma(u'', u)$. Therefore, (s, t) captures (u, v).



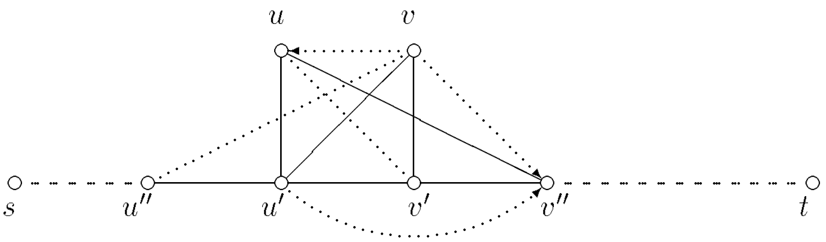
Case 3: $(u', v) \notin \overline{E}$, $(u, v') \in \overline{E}$, and $(u, v'') \in \overline{E}$.

Again, (s, t) forces (u, v) since $(s,t)\Gamma^*(u', v'')\Gamma(u, v'')\Gamma(u, v')\Gamma(u, v)$.



Case 4: $(u', v) \notin \overline{E}$, $(u, v') \in \overline{E}$, and $(u, v'') \notin \overline{E}$.

Here, (s, t) forces (v, u) since $(s,t)\Gamma^*(u', v'')\Gamma(v, v'')\Gamma(v, u)$.



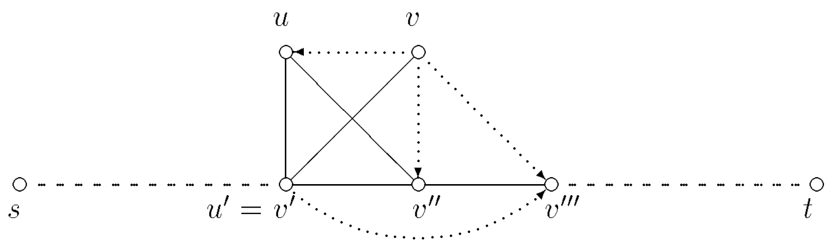
∎

Lemma 4.61 *Let $G = (V, E)$ be a connected graph, and let $P = (s, \ldots, t)$ be a chordless dominating path in G. If P has at least four vertices, then for all $(u, v) \in \overline{E}$* for which *$u' = v'$, either (s, t) forces (u, v); (s, t) forces (v, u); or (s, t) captures (u, v).*

Proof: Since P has at least four vertices, either u' is at least two edges from s in P, or v' is at least two edges from t. We can assume the latter, since the claim for the other possibility follows from symmetry. Again, let us proceed by case.

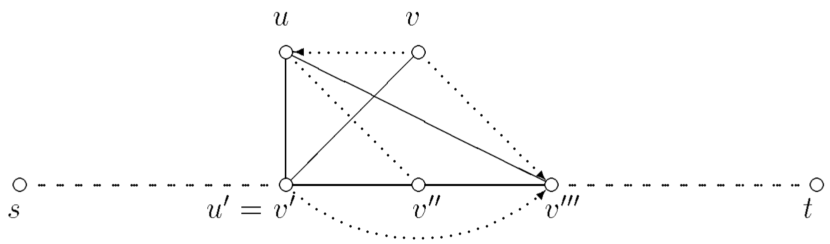
Case 1: $(u, v'') \notin \overline{E}$.

Here $(s,t)\Gamma^*(v', v''')\Gamma(v, v''')\Gamma(v, v'')\Gamma(v, u)$, so (s,t) forces (v,u).



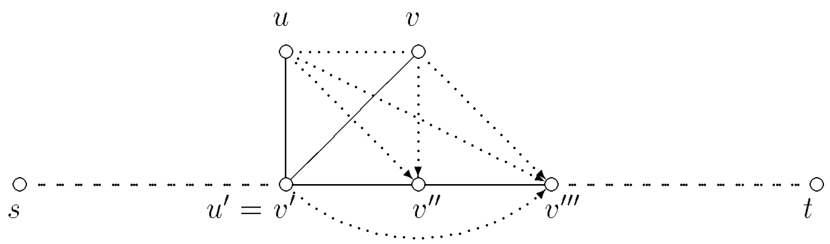
Case 2: $(u, v'') \in \overline{E}$ and $(u, v''') \notin \overline{E}$.

Again, (s,t) forces (v,u) since $(s,t)\Gamma^*(v', v''')\Gamma(v, v''')\Gamma(v, u)$.



Case 3: $(u, v'') \in \overline{E}$ and $(u, v''') \in \overline{E}$.

Clearly, $u \neq u'$ since $(u, v) \in \overline{E}$ but $(u, u') \notin \overline{E}$. Similarly, $v \neq v'$. Therefore, $\{u' = v', u, v, v''\}$ induces a claw in G. Furthermore, $(s,t)\Gamma^*(v', v''')\Gamma(u, v''')\Gamma(u, v'')$ and $(s,t)\Gamma^*(v', v''')\Gamma(v, v''')\Gamma(v, v'')$. Therefore (s,t) captures (u, v).



∎

4.3.3 Implications for Cocomparability Graphs

Let $G = (V, E)$ be a connected cocomparability graph, and let $\vec{G} = (V, A)$ be any transitive orientation of $\overline{E}$. If $(p_0, p_1, \ldots, p_k)$ is a shortest path in G from a source p_0 in $\vec{G}$ to a sink p_k in $\vec{G}$, then P is dominating. For, if v is any undominated vertex, then $(p_0, v)\Gamma(p_1, v)\Gamma \cdots \Gamma(p_k, v)$. So either p_0 is not a source or p_k is not a sink. Furthermore, such a path is chordless, simply because it is a shortest path. This argument establishes that every connected cocomparability graph has a chordless dominating path.

It is instructive to see that this result also follows from the spanning order characterization of cocomparability graphs. A connected cocomparability graph has a spanning order by Theorem 4.3. Since the graph is connected, there is a shortest path from its least vertex to its greatest vertex. This path is dominating by Lemma 4.7.

Chapter 5

Strip Graphs

Recall from Definition 1.3 that a graph $G = (V, E)$ is a *τ-strip graph* if there is a function (a τ-strip realization) $f : V \to \mathbf{R} \times [0, \tau]$ such that $(u, v) \in E$ if and only if $\|f(u) - f(v)\| \leq 1$. A graph is a *strip graph* if it is a $\sqrt{3}/2$-strip graph. Recall that every $\sqrt{3}/2$-strip graph is a cocomparability graph (Theorem 3.7) but that for every $\tau > \sqrt{3}/2$ there is a τ-strip graph that is not a cocomparability graph (Theorem 3.8). In this chapter, we will see how algorithms can exploit a strip-realization for strip graphs. In particular, Section 5.1 develops an algorithm for finding optimal Steiner sets whose run-time is independent of the number of edges.

The rest of this chapter is concerned with characterizing strip graphs; one facet of this is creating a realization given a graph. Section 5.2 characterizes strip graphs as those that have only positive-weight cycles in a corresponding digraph. The algorithm for constructing this digraph assumes knowledge of the y-coordinates of all vertices, as well as the left-to-right order of the realization of nonedges. Section 5.3 immediately applies this characterization to small stars, and shows that, for example, $K_{1,4}$ is a τ-strip graph for some but not all τ. Section 5.4 shows how the characterization distinguishes between indifference graphs and other strip graphs. Specifically, it shows that a strip graph is not an indifference graph precisely if it contains a claw or a square. Finally, Section 5.5 characterizes trees in strip-graphs, showing that all such trees are equivalently caterpillars of degree at most 4.

Let us begin with a brief example. We will see later (Property 5.21) that the bound in this example is tight, that is, that $K_{1,4}$ is not a τ-strip graph for any $\tau \leq \sqrt{5/8}$.

Example 5.1 $K_{1,4}$ is a τ-strip graph for all $\tau > \sqrt{5/8}$.

Proof: Let $\{a, b, c, d\}$ be the independent set of vertices in the star $K_{1,4}$, and let h be the star's "hub". Let d_y be any value satisfying $\sqrt{5/8} < d_y \leq \min\{\sqrt{5/4}, \tau\}$. In particular, if $\sqrt{5/8} < \tau \leq \sqrt{5/4}$, then $d_y = \tau$ will suffice. Define $d_x = \sqrt{(4 - d_y^2)/9}$. Then the assignment $f : V \to \mathbf{R} \times (\sqrt{5/8}, \tau]$ is a strip realization for $K_{1,4}$, where $\tau > \sqrt{5/8}$, and

$$\begin{aligned}
f(a) &= (0, d_y) \\
f(b) &= (d_x, 0) \\
f(c) &= (2d_x, d_y) \\
f(d) &= (3d_x, 0) \\
f(h) &= (3d_x/2, d_y/2)
\end{aligned}$$

Figure 5.1 shows the resulting strip realization when $d_y = \tau = 0.8$. The construction

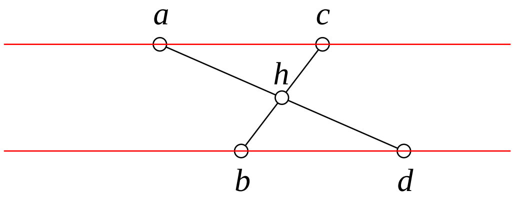


Figure 5.1: $K_{1,4}$ is a 0.8-strip graph.

ensures that $\|f(a) - f(d)\| = 2$ so that $\|f(a) - f(h)\| = \|f(h) - f(d)\| = 1$. To see this, note that

$$\begin{aligned}
\|f(a) - f(d)\|^2 &= (3d_x)^2 + d_y^2 \\
&= 9\left(\frac{4 - d_y^2}{9}\right) + d_y^2 \\
&= 4.
\end{aligned}$$

The construction also ensures that $\|f(b) - f(h)\| = \|f(h) - f(c)\| \leq 1$ since

$$\begin{aligned}
\|f(b) - f(h)\|^2 &= (d_x/2)^2 + d_y^2 \\
&= \frac{4 - d_y^2}{4 \cdot 9} + d_y^2 \\
&= (4 + 37d_y^2)/36 \\
&< 1
\end{aligned}$$

if $d_y^2 < (36 - 4)/37 = 32/37$.

Conversely, the construction ensures that $\|f(a) - f(b)\| = \|f(b) - f(c)\| = \|f(c) - f(d)\| > 1$ since

$$\begin{aligned}
\|f(a) - f(b)\|^2 = &= d_x^2 + d_y^2 \\
&= \frac{4 - d_y^2}{9} + d_y^2 \\
&= \frac{4 + 8d_y^2}{9} \\
&> 1
\end{aligned}$$

if $d_y^2 > (9 - 4)/8 = 5/8$. Finally, the construction ensures that $\|f(a) - f(c)\| = \|f(b) - f(d)\| > 1$ since

$$\begin{aligned}
\|f(a) - f(c)\|^2 = &= (2d_x)^2 \\
&= 4\left(\frac{4 - d_y^2}{9}\right) \\
&> 1
\end{aligned}$$

if $d_y^2 < \frac{9-16}{-4} = 5/4$. ▮

5.1 Exploiting a Geometric Model

Since strip graphs are both unit disk graphs and cocomparability graphs, we can combine algorithms from both classes of graphs. We will see how do this by solving the minimum weight Steiner set problem on strip graphs in $O(V \log V)$ time, given a geometric representation. This result improves on the $O(V \log V + E)$ time bound for the more general cocomparability graphs in Chapter 4. In fact, we will simply implement the cocomparability algorithm using the unit disk graph data structure from Chapter 3.

5.1.1 Review of Cocomparability Results

The following algorithm and terms from Chapter 4 are reproduced here to keep the present discussion self-contained. Let $G = (V, E, w)$ be a graph with nonnegatively weighted vertices. Let $R \subseteq V$ be a "required" set of vertices in G. A *Steiner set* (of G and R) is a set of vertices $S \subseteq V \setminus R$ such that the subgraph induced in G by $S \cup R$ is connected. The weight $w(S)$ of a set of vertices S is defined, as usual, to be the sum of the weights of its elements. Assume without loss of generality that $w(v) = 0$ for all vertices $v \in R$. The *weighted Steiner set problem* is to find a minimum weight Steiner set in such a vertex weighted graph.

Theorem 4.42 [Chapter 4] Given a spanning-ordered cocomparability graph G and required set $R \subseteq V$, algorithm MWSS-OCC returns a minimum weight Steiner set of G and R if and only if it exists.

Theorem 4.43 [Chapter 4] Given any linearly-ordered graph $G = (V, E, <, w)$, algorithm MWSS-OCC runs in $O(V \log V + E)$ time.

5.1.2 Application to Strip Graphs

We want to modify this algorithm to work for strip graphs. The bottleneck in Algorithm MWSS-OCC is finding a lightest path in Step 7. This is precisely the operation that Algorithm MIN-PATH from Chapter 3 was designed to handle for unit disk graphs. A strip graph is of course a unit disk graph where all vertex points are confined to an

Table 5.1: **Algorithm:** MWSS-OCC(G, R) [Minimum Weight Steiner Set for spanning-Ordered Cocomparability Graphs]

Input: A linearly-ordered vertex-weighted graph $G = (V, E, <, w)$ and
a required set of vertices $R \subseteq V$.
Output: A minimum weight Steiner set S,
or a message stating that no Steiner set exists,
or a message stating that G is not a spanning-ordered cocomparability graph.

if R is empty,
  then return the empty set
    halt.
if R is not a subset of a connected component of G,
  then return "No Steiner set exists."
    halt.
$P \leftarrow$ a least weight path from $\min R$ to $\max R$ in G.
$S \leftarrow P \setminus R$
if the subgraph induced by $S + R$ is connected,
  then return S
  else **return** "G is not a spanning-ordered cocomparability graph."

infinite strip τ units thick. Assume the vertices are linearly ordered along the axis of the strip, left to right in this case. This order is a spanning order, by the following lemma. Contrast this lemma with Lemma 3.6, which states that the *compatible orientation* (as opposed to linear order) associated with any strip realization of a graph is a transitive orientation.

Lemma 5.2 *Every* left-to-right-ordered *strip graph is a* spanning-ordered *cocomparability graph.*

Proof: Let $f : V \to [0, \tau]$ be a τ-strip realization of a graph $G = (V, E)$, where $\tau \in [0, \sqrt{3}/2]$. Let u, v, and w be three vertices where u and w are adjacent, and $x_f(u) \leq x_f(v) \leq x_f(w)$. Since $f(u)$ and $f(w)$ are at most unit distance apart, their projections, $x_f(u)$ and $x_f(w)$, on the x-axis are at most unit distance apart, too. Since $x_f(v)$ lies between $x_f(u)$ and $x_f(w)$, it must lie within half a unit of either $x_f(u)$ or $x_f(w)$, say $x_f(u)$, without loss of generality. Therefore the square of the distance between $f(u)$ and $f(v)$ is given by

$$\begin{aligned}
(x_f(u) - x_f(v))^2 + (y_f(u) - y_f(v))^2 &\leq (1/2)^2 + \tau^2 \\
&\leq (1/2)^2 + (\sqrt{3}/2)^2 \\
&= 1.
\end{aligned}$$

That is, u and v are adjacent, and the vertex order is a spanning order. ∎

Create Algorithm MWSS-S (Minimum Weight Steiner Set on Strip Graphs) by removing Steps 9 through 11 from Algorithm MWSS-OCC, and by inserting Step 0, which sorts the strip graph vertices from left to right (i.e., by x-coordinate).

Theorem 5.3 *Given a strip graph G, represented as a set of points in a strip, and required vertices R, Algorithm MWSS-S returns a minimum weight Steiner set if and only if it exists, and runs in $O(V \log V)$ time.*

Proof: Correctness follows from Theorem 4.42 and Lemma 5.2. Steps 9 through 11 are not required by Lemma 5.2.

Represent all sets as arrays with V elements; linked lists and arrays are easily interconverted where required in $O(V)$ time. Step 0 takes $O(V \log V)$ time, and Step 1 takes $O(V)$ time.

Step 4 can be implemented by marking all vertices in the same connected component as the first vertex in R using a single phase of depth first search in $O(V \log V)$ time (Theorem 3.16). It then suffices to check, in $O(R)$ time, that all vertices in R are marked. Step 4 therefore executes in $O(V \log V)$ time.

The extreme vertices $\min R$ and $\max R$ can easily be found in $O(R) = O(V)$ time. Step 7 uses algorithm MIN-PATH (Table 3.3) to find a least weight path between $\min R$ and $\max R$, which takes $O(V \log V)$ time by Theorem 3.14.

Step 8, setting $S = P \setminus R$, can be computed in $O(V)$ time by filling in the array for S. ▮

5.2 Characterization Via Cycles in Weighted Digraphs

Let $G = (V, E)$ be a τ-strip graph, and let $f : V \to \mathbf{R} \times [0, \tau]$ be a realization of G. Let us suppose that we know the y-coordinates of $f(V)$. Then we know something about the x-dimension between pairs of vertices, as clarified by the following definition and its property.

Definition 5.4 For every pair of vertices $u, v \in V$, call $\alpha_{u,v} = \sqrt{1 - (y_f(u) - y_f(v))^2}$ the *critical x-dimension* corresponding to the pair. Note that, if $0 \leq \tau \leq \sqrt{3}/2$, then $1/2 \leq \alpha_{u,v} \leq 1$ for all pairs u, v. ▮

Property 5.5 *Let $G = (V, E)$ be a τ-strip graph, and let $f : V \to \mathbf{R} \times [0, \tau]$ be a realization of G. For every distinct pair $u, v \in V$, $(u, v) \in E$ if and only if*

$$x_f(u) - x_f(v) \leq \alpha_{uv} \tag{5.17}$$

and

$$x_f(v) - x_f(u) \leq \alpha_{uv}. \tag{5.18}$$

Proof: By Definition 5.4,

$$\begin{aligned} (u, v) \in E \quad & \text{if and only if} \quad \|f(u) - f(v)\| \leq 1, \\ & \text{if and only if} \quad |x_f(u) - x_f(v)|^2 + |y_f(u) - y_f(v)|^2 \leq 1, \\ & \text{if and only if} \quad |x_f(u) - x_f(v)| \leq \alpha_{uv}. \end{aligned}$$

∎

Suppose also that we know the left-to-right order of the endpoints of each nonedge. Note that this is weaker than knowing the left-to-right order of all of the vertices. That is, let $(V, \vec{E})$ be the strict partial order where $(u, v) \in \vec{E}$ if and only if $(u, v) \in \overline{E}$ and $x_f(u) < x_f(v)$. Note that $\vec{E}$ is an orientation of $\overline{E}$.

Property 5.6 *For every pair $u, v \in V$, $(u, v) \in \vec{E}$ if and only if*

$$x_f(v) - x_f(u) > \alpha_{uv}$$

equivalently, if and only if

$$x_f(u) - x_f(v) < -\alpha_{uv}. \tag{5.19}$$

Proof: By Property 5.5, $(u, v) \in \overline{E}$ if and only if $|x_f(u) - x_f(v)| > \alpha_{uv}$. ∎

Note that inequalities 5.17 through 5.19 generate a system of

$$|E| + |E| + (|V|^2 - |V| - |E|) = |V|^2 - |V| + |E|$$

difference constraints. Let us make some of these notions more formal.

Definition 5.7 A *(τ) levelled graph* $G = (V, E, l)$ is a graph $G = (V, E)$ and a *levelling function* $l : V \to [0, \tau]$ that maps vertices to y-coordinates. ∎

Definition 5.8 A levelled graph $G = (V, E, l)$ is a *levelled τ-strip graph* if there exists a τ-strip realization $f : V \to \mathbf{R} \times [0, \tau]$ for G such that $y_f(v) = l(v)$, for every vertex $v \in V$. ∎

Recall from Chapter 1 that an *oriented graph* is a directed graph $G = (V, A)$ where A is asymmetric. Also from Chapter 1, an *orientation* of an undirected graph $G = (V, E)$ is an oriented subgraph $H = (V, A)$ where $E = A + A^{-1}$ (the arc set A is also called an *orientation* of the edge set E).

Definition 5.9 A *complement oriented graph* $G = (V, E, \vec{E})$ is a graph $G = (V, E)$ and an orientation $\vec{G} = (V, \vec{E})$ of the complement $G = (V, \overline{E})$. ∎

Definition 5.10 A complement oriented graph $G = (V, E, \vec{E})$ is a *complement oriented τ-strip graph* if there exists a τ-strip realization $f : V \to \mathbf{R} \times [0, \tau]$ for G such that $x_f(v) < x_f(v)$ whenever $(u, v) \in \vec{E}$. ∎

By the above discussion, a strip graph can be levelled and complement oriented such that Inequalities 5.17 through 5.19 are satisfied. Conversely, if the inequalities corresponding to a levelled, complement oriented graph are satisfied by some assignment, then that assignment, together with the levelling, is also a strip realization. We have established the following theorem:

Theorem 5.11 *Given a levelled, complement oriented graph $G = (V, E, l, \vec{E})$, let B be the corresponding system of $\Theta(V^2)$ difference constraints (5.17 through 5.19). A function f is a strip realization for G if and only if $\{x_u : f(u) = (x_u, l(u))\}$ is a feasible solution for B.*

Corollary 5.11.1 *A graph is a τ-strip graph if and only if it can be τ-levelled and oriented such that the corresponding system of difference constraints has a solution.*

5.2.1 Solving Systems of Difference Constraints

According to Theorem 5.11, we can recognize levelled, complement oriented, τ-strip graphs by generating and solving a system of inequalities. This section develops an algorithm that solves systems of mixed difference constraints such as Inequalities 5.17 through 5.19. We will begin by describing Bellman's solution of nonstrict difference constraints [Bel58], and then extend his methodology to include strict inequalities. The extension is self-contained, so Bellman's results will be described but not proved.

Bellman's Solution

The following terminology follows [CLR90]. A *system of difference constraints* is a set of m inequalities (on n variables) of the form $x_j - x_i \leq b_k$, where $1 \leq i, j \leq n$ and $1 \leq k \leq m$. A vector of values $\mathbf{x} = (x_1, \ldots, x_n)$ is a *feasible solution* of a system of difference constraints if it satisfies the system. If $x_i - x_j \leq b_k$ is a constraint in the system, then all other constraints $x_i - x_j \leq b_l$, where $b_l \geq b_k$, are redundant in terms of feasible solutions. Therefore we may assume that any system has at most one constraint of the form $x_j - x_i \leq b_k$, and at most one constraint of the form $x_i - x_j \leq b_l$, for each pair i, j. The corresponding *constraint digraph* is a weighted directed graph $D = (V, A, w)$

where:

$$\begin{aligned}
V &= \{v_0, v_1, \ldots, v_n\}, \\
A &= \{(v_i, v_j) : x_j - x_i \le b_k \ \text{ is a constraint}\} \cup \{(v_0, v_i) : 1 \le i \le n\} \\
w(v_i, v_j) &= b_k \ \text{ if } x_j - x_i \le b_k \ \text{ is a constraint} \\
w(v_0, v_i) &= 0 \ \text{ where } \ 1 \le i \le n
\end{aligned}$$

The weight of a path (or cycle) in D is the sum of the weights of its arcs. Let $\delta(u, v)$ denote the least weight of any path from u to v in D.

Theorem 5.12 ([CLR90], page 542) *Given a system of difference constraints, let D be the corresponding constraint digraph. If D contains no negative-weight cycles, then*

$$x = (\delta(v_0, v_1), \delta(v_0, v_2), \ldots, \delta(v_0, v_n))$$

is a feasible solution for the system. If D contains a negative-weight cycle, then there is no feasible solution for the system.

Adding Strict Inequalities

Let us begin by generalizing the notion of difference constraints. Let $x_j - x_i \tilde{<} b_k$ denote a constraint of the form $x_j - x_i \le b_k$ or $x_j - x_i < b_k$. A *system of mixed difference constraints* is a set of m inequalities (on n variables) of the form $x_j - x_i \tilde{<} b_k$, where $1 \le i, j \le n$ and $1 \le k \le m$. Again, we can assume that there is at most one constraint $x_j - x_i \tilde{<} b_k$ and at most one $x_i - x_j \tilde{<} b_l$ for each pair i, j. The corresponding *mixed constraint digraph* is a weighted directed graph $D = (V, A, w)$ where:

$$V = \{v_0, v_1, \ldots, v_n\},$$

$$\begin{aligned} A &= \{(v_i, v_j) : x_j - x_i \tilde{<} b_k \text{ is a constraint}\} \cup \\ &\quad \{(v_0, v_i) : 1 \leq i \leq n\} \\ w(v_i, v_j) &= b_k \text{ if } x_j - x_i \tilde{<} b_k \text{ is a constraint} \\ w(v_0, v_i) &= 0 \text{ where } 1 \leq i \leq n \end{aligned}$$

A *loose constraint* is one with a strict inequality; the other constraints are *tight*. A *loose arc* is one corresponding to a loose constraint, and a *loose cycle* is one containing a loose arc.

Lemma 5.13 *Given a system B of mixed difference constraints, let D be the corresponding mixed constraint digraph. If B has a feasible solution, then D contains no negative-weight cycles, and all loose cycles have positive weight.*

Proof: Suppose that B has a feasible solution, and let $C = (c_0, c_1, \ldots, c_{p-1}, c_0)$ be any directed cycle in D with $p > 1$. Let $x_{i+1} - x_i \tilde{<} b_{i,i+1}$ be the constraint corresponding to (c_i, c_{i+1}). By construction and the feasibility of the solution, $w(c_i, c_{i+1}) = b_{i,i+1} \tilde{>} x_{i+1} - x_i$. Therefore, the weight of the cycle satisfies

$$\begin{aligned} \sum_{i=0}^{p-1} w(c_i, c_{i+1}) &= \sum_{i=0}^{p-1} b_{i,i+1} \\ &\geq \sum_{i=0}^{p-1} x_{i+1} - x_i \\ &= -x_0 + x_0 \\ &= 0, \end{aligned}$$

where subscript arithmetic is modulo p. That is, every cycle has nonnegative weight. If C contains at least one loose arc, then the inequality is strict. That is, every loose cycle has positive weight. ∎

The proof and extended statement of the converse of Lemma 5.13 require the following notions. Let $D = (V, A, w)$ be the weighted digraph corresponding to a set B of mixed difference constraints. Our goal is to create a digraph for a system of tight difference constraints that has a feasible solution if and only if B has a feasible solution. To do so, we first convert loose arcs to tight arcs by inserting a necessary "space" λ into the constraints. If D has no loose cycles, define λ to be any positive constant. Otherwise, let C_0 be a least-weight loose cycle, and let λ be any value that satisifies

$$0 < \lambda \leq \frac{w(C_0)}{|V| - 1}. \tag{5.20}$$

Now create a new directed graph D' from D by subtracting λ from the weight of every loose arc.

Definition 5.14 The *tightened constraint digraph* corresponding to $D = (V, A, w)$ is the weighted digraph $D' = (V, A, w')$ where

$$w'(u, v) = \begin{cases} w(u, v) - \lambda & \text{if } (u, v) \text{ is a loose arc} \\ w(u, v) & \text{if } (u, v) \text{ is a tight arc.} \end{cases}$$

▮

Theorem 5.15 *Given a system B of mixed difference constraints, let D be the corresponding mixed constraint digraph, and let D' be the corresponding tightened constraint digraph. If D contains no negative-weight cycles, and all loose cycles have positive weight, then $\mathbf{x} = \{x_u : u \in V \text{ and } x_u = \delta'(v_0, u)\}$ is a feasible solution of B. If D contains a negative-weight cycle, or a loose cycle with non-positive weight, then there is no feasible solution for the system B.*

Proof: The tightened constraint digraph D' has no negative-weight cycles. To see this, let C be a cycle in D'. If C is not loose, then it has the same nonnegative weight in D' that it has in D. Otherwise, assume it has j loose arcs. Then, since $j \leq |V| - 1$,

$$w'(C) \;=\; w(C) - j\lambda$$

$$\begin{aligned} &= w(C) - j\frac{w(C_0)}{|V|-1} \\ &\geq w(C_0) - w(C_0) \\ &= 0. \end{aligned}$$

Therefore the least path weight $\delta'(u,v)$ from u to v in D' is well defined.

Now let (u,v) be any arc in D'. Then $\delta'(v_0,v) \leq \delta'(v_0,u) + w'(u,v)$, for otherwise the path from v_0 through u to v (with weight $\delta'(v_0,u) + w'(u,v)$) would be lighter than the lightest path from v_0 to v (with weight $\delta'(v_0,v)$). It follows that

$$x_v - x_u = \delta'(v_0,v) - \delta'(v_0,u) \leq w'(u,v).$$

Therefore $x_v - x_u < w(u,v)$ if (u,v) is loose, and $x_v - x_u \leq w(u,v)$ otherwise. It follows that $\mathbf{x}$ is a feasible solution of B. The second claim follows immediately from Lemma 5.13. ∎

5.2.2 Constraint Digraphs Corresponding to τ-Strip Graphs

For strip graphs, it is convenient to go directly to the corresponding constraint digraph by combining Theorem 5.11 and Theorem 5.15 as follows.

Definition 5.16 The *(mixed constraint) digraph corresponding to a levelled, complement oriented graph* $G = (V, E, l, \vec{E})$ is a weighted, directed graph $D = (V_D, A, w)$ where:

$$\begin{aligned} V_D &= V \cup \{v_0\} \\ A &= A^+ \cup A^- \cup \{(v_0,v) : v \in V\} \\ A^+ &= \{(u,v) : (u,v) \in E\} \\ A^- &= \{(u,v) : (v,u) \in \vec{E}\} \end{aligned}$$

The *weight* $w(u,v)$ of each directed arc (u,v) in D is:

$$w(u,v) = \begin{cases} \alpha_{uv} & \text{if } (u,v) \in A^+ \\ -\alpha_{uv} & \text{if } (u,v) \in A^- \\ 0 & \text{if } u = v_0 \end{cases}$$

where $\alpha_{uv} = \sqrt{1 - (l(u) - l(v))^2}$. See Figure 5.2 for a small example. ∎

(a) (b)

Figure 5.2: The weighted digraph (b) corresponding to the star $K_{1,4}$ (a). The eight arcs with positive weight in Figure (b) are shown as solid arrows. Note that there are two solid arrows between every pair of vertices that are adjacent in Figure (a). The six arcs with negative weight in Figure (b) are shown as dotted arrows. Note that there is exactly one dotted arrow between every distinct pair of vertices that are not adjacent in Figure (a). The solid arrows all have weight $\alpha' = \sqrt{1 - (\tau/2)^2}$. The horizontal dotted arrows all have weight $-1 = -\sqrt{1 - 0^2}$. The diagonal dotted arrows all have weight $-\alpha = -\sqrt{1 - \tau^2}$.

Arcs in A^+ are called *positive*, and arcs in A^- are called *negative*. Note that, for each edge in E, there are two oppositely directed arcs in A^+ with the same positive weight. Note also that the negative arcs A^- in D are exactly the loose arcs. So, if C is a cycle in D with no loose arcs, then it must have positive weight.

Theorem 5.17 *Given a τ-levelled, complement oriented graph G, let D be the corresponding mixed constraint digraph, and let D' be the corresponding tightened constraint digraph. If every cycle in D has positive weight, then $f : V \to \mathbf{R}^2$, where*

$f(u) = (\delta'(u), l(u))$ for all $u \in V$, is a strip-realization of G. If D contains a nonpositive-weight cycle, then there is no strip-realization with the given levelling and complement orientation.

Proof: Let B be the corresponding system of $\Theta(V^2)$ difference constraints (5.17 through 5.19). By Theorem 5.11, f is a strip realization for G if and only if $\{x_u : f(u) = (x_u, y(u))\}$ is a feasible solution for B. The digraph D corresponding to the levelled and complement oriented graph G also corresponds to this system of constraints B. If every cycle in D is positive, $\{x_u : x_u = \delta'(v_0, u)$ and $u \in V\}$ is a feasible solution of B by Theorem 5.15.

Finally, if D contains a nonpositive-weight cycle C, then C must be loose. By Theorem 5.15, there is no feasible solution for B, and the theorem follows by Theorem 5.11. ∎

Corollary 5.17.1 *A graph is a τ-strip graph if and only if it can be τ-levelled and complement oriented such that every cycle in its corresponding digraph has positive weight.*

5.2.3 An $O(V^3)$ Algorithm for Laying Out Levelled, Complement Oriented, Strip Graphs

Theorem 5.17 promises an algorithm to lay out a strip graph once it has been levelled and complement oriented. Algorithm STRIP-LAYOUT is a straightforward implementation of the ideas behind the theorem.

Theorem 5.18 *Algorithm STRIP-LAYOUT recognizes and produces a τ-strip realization for a levelled, complement oriented graph $G = (V, E, l, \vec{E})$ in $O(V^3)$ time.*

Proof: Given a levelled, complement oriented graph $G = (V, E, l, \vec{E})$, Step 1 can build an $n \times n$ matrix representation of the corresponding mixed constraint digraph $D =$

Table 5.2: **Algorithm:** STRIP-LAYOUT(G) [Levelled, Complement Oriented, Strip Graph Recognition]

Input: A levelled, complement oriented graph $G = (V, E, l, \vec{E})$ and a real number τ.

Output: A shortest least-weight cycle in D and a realization $f : V \to \mathbf{R} \times [0, \tau]$ if G is a τ-strip graph, or evidence (a negative cycle in D) that G is not a τ-strip graph.

$V_D \leftarrow V \cup \{v_0\}$
$D = (V_D, A, w) \leftarrow$ the digraph corresponding to G.
$\{\delta(u, v) : u, v \in V_D\} \leftarrow$ FLOYD-WARSHALL(D)
$w_0 = \min\{\delta(v, v) : v \in V\}$
return the cycle in D corresponding to w_0.
if $w_0 > 0$
    then $\lambda \leftarrow w_0/(|V| - 1)$
            $D' = (V_D, A, w') \leftarrow$ the tightened digraph corresponding to G.
            $\{\delta'(u, v) : u, v \in V_D\} \leftarrow$ FLOYD-WARSHALL(D')
            return function f, where $f(v) = (\delta'(v_0, v), l(v))$ for all $v \in V$.

(V_D, A, w) in $O(V_D^2) = O(V^2)$ time. Let us assume that $w(v,v) = \infty$ for all $v \in V_D$. Step 2 can then run the dynamic programming Floyd-Warshall all-pairs shortest-paths algorithm ([CLR90] pages 558–565) on $D(V)$ in $O(V^3)$ time. Step 3 can then set $w_0 = \min\{\delta(v,v) : v \in V\}$ from the result of Step 2 in $O(V)$ additional time.

Implement the Floyd-Warshall algorithm to maintain, for all pairs u, v, the predecessor $p(u,v)$ of v in the (computed) shortest path from u to v. This does does not increase the asymptotic run time, but allows Step 4 to extract the cycle corresponding to w_0 in linear time ([Tar83] pages 94–95). The Floyd-Warshall algorithm has the property that this cycle is negative if $w_0 < 0$, in which case G has no layout compatible with its levelling and complement orientation, again by Theorem 5.17.

If $w_0 > 0$, then D has no negative cycles, and $\delta(u,v)$ correctly represents the least path-weight from u to v for all such pairs. In particular, w_0 is the weight of a least-weight cycle in D, and λ in Step 6 satisfies Inequality 5.20. Step 7 can now construct the tightened constraint digraph $D' = (V_D, A, w')$ corresponding to D in $O(V_D + A) = O(V^2)$ time. Finally, Step 8 again[1] executes the Floyd-Warshall algorithm in $O(V^3)$ time. The entire recognition and layout algorithm for levelled, complement oriented strip graphs therefore runs in $O(V^3)$ time. ∎

5.3 Stars in Strip-Graphs

As an example of the weighted cycle mechanism of the previous section, this section shows which stars are τ-strip graphs as the thickness τ of the strip varies.

Property 5.19 *The star $K_{1,5}$ is not a τ-strip graph for any $\tau \leq \sqrt{3}/2$.*

[1]Since the time-critical step in this algorithm is Step 2, it suffices for Step 8 to execute in $O(V^3)$ time. If you have a faster way to find the lightest cycle in Step 2, then you should also use a faster algorithm for Step 8. One such algorithm is the Bellman-Ford algorithm for the single-source least path-weight problem ([CLR90] pages 532–544). The Bellman-Ford algorithm runs in $O(VA) = O(V^3)$ time, which is faster for sparse graphs.

Proof: Consider any levelling and complement orientation of $K_{1,5}$. Its 5 independent vertices are then linearly ordered, say $a < b < c < d < e$. Then $C = (h, e, d, c, b, a, h)$ is a cycle in the corresponding digraph, where h is the "hub" of the star. This cycle has weight

$$\begin{aligned} w(C) &= \alpha_{he} - \alpha_{ed} - \alpha_{dc} - \alpha_{cb} - \alpha_{ba} + \alpha_{ah} \\ &\leq 1 - 1/2 - 1/2 - 1/2 - 1/2 + 1 \\ &= 0 \end{aligned}$$

since, for all arcs (u, v), $\sqrt{1-\tau^2} \leq \alpha_{uv} \leq 1$ and $\sqrt{1-\tau^2} = \sqrt{1-(3/4)} = 1/2$. Hence $K_{1,5}$ is not a strip graph by Theorem 5.17. ▮

Property 5.20 *The star $K_{1,4}$ is not a τ-strip graph for any $\tau \leq \sqrt{5/9}$.*

Proof: Again, consider any levelling and complement orientation of $K_{1,4}$ such that $C = (h, d, c, b, a, h)$ is a cycle in the corresponding digraph where $a < b < c < d$ are the independent vertices and h is the hub. This cycle has nonpositive weight

$$\begin{aligned} w(C) &= \alpha_{hd} - \alpha_{dc} - \alpha_{cb} - \alpha_{ba} + \alpha_{ah} \\ &\leq 1 - 2/3 - 2/3 - 2/3 + 1 \\ &= 0 \end{aligned}$$

since, for all arcs (u, v), $\sqrt{1-\tau^2} \leq \alpha_{uv} \leq 1$ and $\sqrt{1-\tau^2} = \sqrt{1-(5/9)} = 2/3$. ▮

We saw in Example 5.1 that $K_{1,4}$ is a strip graph for all strips of thickness greater than $\sqrt{5/8}$, and in the previous property, that $K_{1,4}$ is not embedable in strips thinner than $\sqrt{5/9}$. Is one of these bounds tight? As it turns out, the $\tau > \sqrt{5/8}$ bound is tight. The weighted cycle mechanism from the previous section does not provide enough power to prove this by itself. We must therefore resort to more *ad hoc* arguments.

Property 5.21 *The star $K_{1,4}$ is not a τ-strip graph for any $\tau \leq \sqrt{5/8}$.*

Proof: The diameter of any set of points realizing $K_{1,4}$ must be at most 2 since every pair of vertices is joined by a path of length at most 2. In particular, any realization of the four (independent) leaves must have diameter at most 2. However, we will see that any $\sqrt{5/8}$-realization of four independent vertices has diameter greater than 2.

To this end, let $\{a, b, c, d\}$ be a set of points that realizes four independent vertices in a $\sqrt{5/8}$-strip. Without loss of generality, we may assume that $x(a) < x(b) < x(c) < x(d)$, so that $\|a - d\|$ is the diameter. Therefore $x(b) - x(a) > \sqrt{1 - (5/8)} = \sqrt{3/8} > 1/2$, and this relation also holds for $x(c) - x(b)$ and $x(d) - x(c)$. It follows that $x(d) - x(a) > 3\sqrt{3/8}$. We may also assume that a and d minimize $\|a - d\|$, and that b and c maximize $\|b - c\|$ (over all possible realizations). Then all points in $\{a, b, c, d\}$ lie on the boundaries of the strip.

To see this, suppose first that a does not lie on the boundary, as shown in Figure 5.3. Then we can perturb a such that the resulting realization has a smaller diameter $\|a - d\|$,

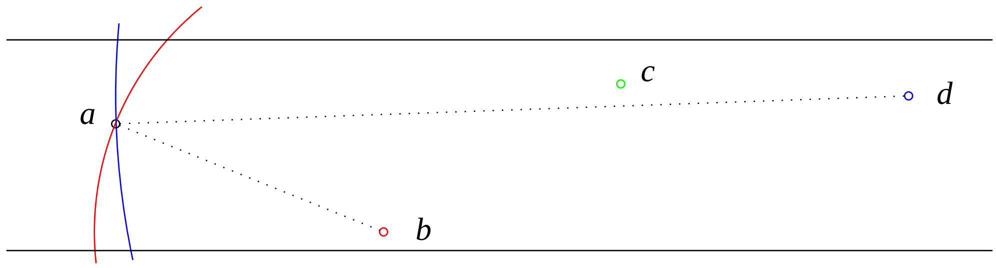


Figure 5.3: If point a is not on the boundary, then rotate it about b to decrease $\|a - d\|$. The figure shows two arcs, centered at b and d, and passing through a. In the figure, b lies below segment ad so rotate a clockwise along the arc about b. The arc about d delimits the points that are farther from d from those that are closer to d.

as follows. If b lies on or below the line segment ad, as shown in Figure 5.3, then rotate a clockwise about b, thereby moving a closer to d. This process does not change the distance $\|a - b\| > 1$, so that $x(b) - x(a)$ remains greater than $\sqrt{3/8}$. It follows that

$x(c) - x(a) = (x(c) - x(b)) + (x(b) - x(a)) > 1/2 + 1/2 = 1$, so that a and c remain independent. By a symmetric argument (reflect the points about the horizontal), if b lies above segment ad, then decrease the diameter by rotating a counterclockwise about b. Similarly, if d does not lie on the boundary, then decrease the diameter by rotating it about c, either clockwise or counterclockwise, depending on whether c lies above or below segment ad, respectively.

Now suppose that b does not lie on the boundary. If b lies on or below the line segment ac, as shown in Figure 5.4, then rotate b clockwise about a, thereby moving b farther from

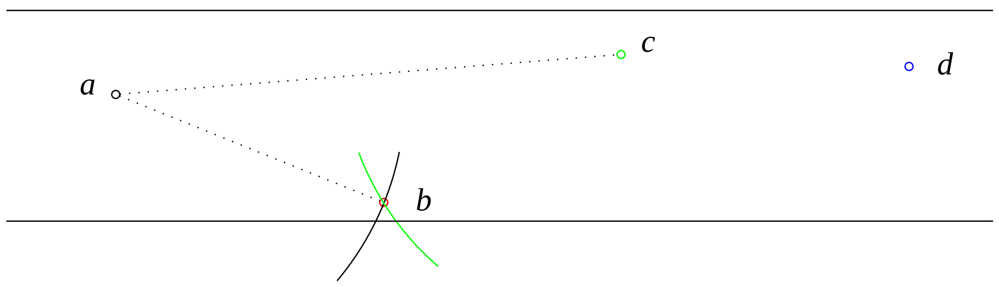


Figure 5.4: If point b is not on the boundary, then rotate it about a to increase $\|b - c\|$. The figure shows two arcs, centered at a and c, and passing through b. In the figure, b lies below segment ac, so rotate b clockwise along the arc about a. The arc about c delimits the points that are farther from c from those that are closer to c.

c. Again, this process does not change the distance $\|a - b\| > 1$, so that a and b remain independent. Also $x(d) - x(b) = (x(d) - x(c)) + (x(c) - x(b)) > 1/2 + 1/2 = 1$, so that b and d remain independent. By a symmetric argument (reflect the points about the horizontal again), if b lies above segment ac, then increase $\|b - c\|$ by rotating b counterclockwise about a. Similarly, if c does not lie on the boundary, then increase $\|b - c\|$ by rotating it about d, either clockwise or counterclockwise, depending on whether c lies below or above segment bd, respectively.

Now that we know that $\{a, b, c, d\}$ lies on the boundary, we must analyze the possibilities. Suppose a and d lie on different levels. Then the distance between them satisfies

$$\|a - d\|^2 > (3\sqrt{3/8})^2 + (5/8) = 2^2$$

so that the diameter of the point set is greater than 2. On the other hand, suppose that a and d lie on the same level. Then the points in at least one pair (a, b), (b, c), or (c, d) must also lie on the same level (as each other). The x-distance between this pair is the same as the distance between the pair, which must be greater than 1. Therefore the x-distance between a and d satisfies:

$$x(d) - x(a) > (2\sqrt{3/8}) + 1 > 2.22$$

so the diameter is again greater than 2. ∎

The following theorem summarizes Example 5.1 and Property 5.21.

Theorem 5.22 *The star $K_{1,4}$ is a τ-strip graph precisely when $\tau > \sqrt{5/8}$.*

Property 5.23 *The star $K_{1,4}$ is not an L_1 τ-strip graph for any $\tau \leq 1/2$.*

Proof: This proof is analogous and nearly identical to that of Property 5.21, so it is worded as similarly as possible to emphasize this analogy. Again, we will show that any four independent points $\{a, b, c, d\}$ under the L_1 metric in a strip at most 1/2-unit thick has diameter greater than 2. Again assume without loss of generality that $x(a) < x(b) < x(c) < x(d)$. Therefore $x(b) - x(a) > 1 - (1/2) = 1/2$, and this relation also holds for $x(c) - x(b)$ and $x(d) - x(c)$. It follows that $x(d) - x(a) > 3(1/2)$. This time, assume that a and d minimize the *x-span* $x(d) - x(a)$, and that b and c maximize $x(c) - x(b)$ (over all possible realizations). Then all points in $\{a, b, c, d\}$ lie on the boundary of the strip.

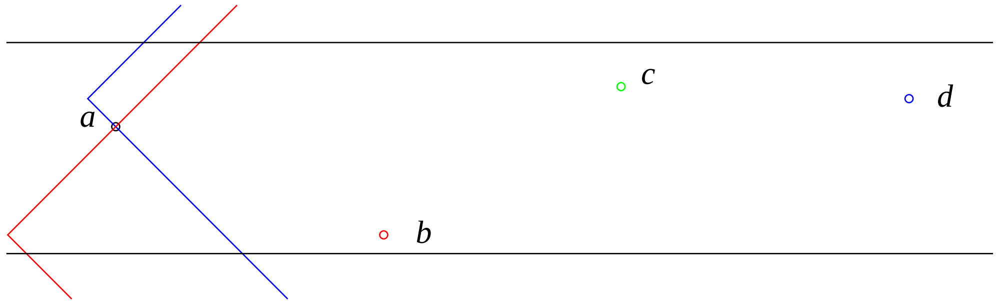


Figure 5.5: If point a is not on the boundary, then rotate it about b to decrease $x(d)-x(a)$ while not decreasing $\|a-d\|_1$. The figure shows two arcs, centered at b and d, and passing through a. In the figure, b lies below a so rotate a clockwise along the arc about b. The arc about d delimits the points that are farther from d from those that are closer to d.

To see this, suppose first that a does not lie on the boundary, as shown in Figure 5.5. Then we can perturb a such that the resulting realization has a smaller value $x(d)-x(a)$, as follows. If b lies at or below $y(a)$, as shown in Figure 5.5, then rotate a clockwise about b, thereby moving a closer to d in the x-dimension. Note that this may decrease the L_1 distance between a and d, or it may leave it unchanged. However, this process does not change the distance $\|a-b\|_1 > 1$, so that $x(b)-x(a)$ remains greater than $1/2$. It follows that $x(c)-x(a) = x(c)-x(b)+(x(b)-x(a)) > 1/2+1/2 = 1$, so that a and c remain independent. By a symmetric argument (reflect the points about the horizontal), if b lies above a, then decrease the x-span by rotating a counterclockwise about b. Similarly, if d does not lie on the boundary, then decrease the x-span by rotating it about c, either clockwise or counterclockwise, depending on whether c lies above or below d, respectively.

Now suppose that b does not lie on the boundary. If b lies at or below $y(a)$, as shown in Figure 5.6, then rotate b clockwise about a, thereby moving b farther from c. Note that this may increase the L_1 distance between b and c, or it may leave it unchanged. Again, this process does not change the distance $\|a-b\| > 1$, so that a and b remain independent. Also $x(d)-x(b) = x(d)-x(c)+(x(c)-x(b)) > 1/2+1/2 = 1$, so that b and d remain independent. By a symmetric argument (reflect the points about the horizontal

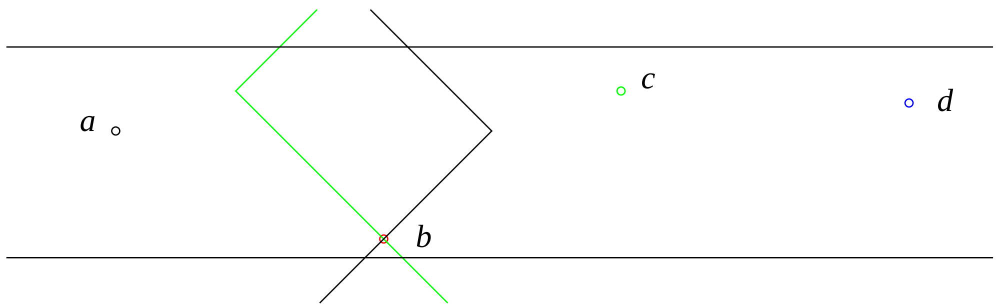


Figure 5.6: If point b is not on the boundary, then rotate it about a to increase $x(c)-x(b)$. The figure shows two arcs, centered at a and c, and passing through b. In the figure, b lies below segment ac, so rotate b clockwise along the arc about a. The arc about c delimits the points that are farther from c from those that are closer to c.

again), if b lies above point d, then increase $x(c) - x(b)$ by rotating b counterclockwise about a. Similarly, if c does not lie on the boundary, then increase $x(c)-x(b)$ by rotating it about d, either clockwise or counterclockwise, depending on whether c lies below or above point d, respectively.

Now that we know that $\{a, b, c, d\}$ lies on the boundary again, we must analyze the possibilities. Suppose a and d lie on different levels. Then the distance between them satisfies

$$\|a - d\|_1 > 3(1/2) + (1/2) = 2$$

so that the diameter of the point set is greater than 2. On the other hand, suppose that a and d lie on the same level. Then the points in at least one pair, (a, b), (b, c), or (c, d) must also lie on the same level (as each other). The x-distance between this pair is the same as the distance between the pair, which must be greater than 1. That is, the x-distance between a and d satisfies:

$$x(d) - x(a) > 2(1/2) + 1 = 2$$

so the diameter is again greater than 2. ∎

Theorem 5.24 *The star $K_{1,3}$ (claw) is a τ-strip graph precisely when $\tau > 0$.*

Proof: The claw is easy to realize if $\tau > 0$ as shown in Figure 5.7. However it is not

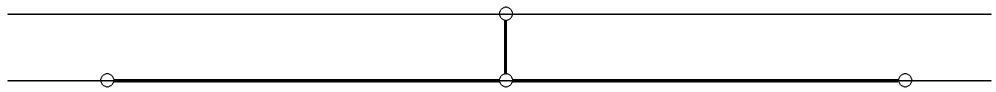


Figure 5.7: A claw in a thin strip

a 0-strip (indifference) graph by Theorem 2.3.4. We can prove this directly, using the weighted cycle mechanism. Consider any orientation of $K_{1,3}$ such that $C = (h, c, b, a, h)$ is a cycle in the corresponding digraph where $a < b < c$ are the independent vertices and h is the hub. This cycle has nonpositive weight

$$\begin{aligned} w(C) &= \alpha_{hc} - \alpha_{cb} - \alpha_{ba} + \alpha_{ah} \\ &= 1 - 1 - 1 + 1 \\ &= 0 \end{aligned}$$

since, for all arcs (u, v), $\alpha_{u,v} = \sqrt{1 - \tau^2} = \sqrt{1 - 0} = 1$. ▮

5.4 Distinguishing Strip Graphs and Indifference Graphs

The weighted digraph corresponding to a τ-strip was defined previously (Definition 5.16). This section continues to examine cycles in these weighted digraphs. Let us say that a strip graph is *proper* if it is not a strip graph for $\tau = 0$, that is, if it is not an indifference graph. This section defines the notion of a *dangerous cycle*, and shows that there is a dangerous cycle in every weighted digraph that corresponds to a proper strip graph. We will see that if such a digraph has a dangerous cycle, then it has one with four arcs. Consequently proper strip graphs contain either a square or a claw, since a dangerous 4-cycle must correspond to one or the other.

A graph or digraph is *signed* if there is a positive (+) or negative (−) sign associated with each edge or arc of the graph. In this section, it is often easier to think of our weighted digraphs as signed, depending on whether the arc weight is positive or negative.

Definition 5.25 A cycle in a signed digraph is *dangerous* if it contains at least as many negative arcs as positive arcs, that is, if $\frac{n}{p} \geq 1$, where n is the number of negative cycle arcs and p is the number of positive cycle arcs. ▮

5.4.1 Dangerous 4-Cycles

Lemma 5.26 *Let D be the digraph corresponding to a levelled, complement oriented graph G. If C is a positive, dangerous cycle in D, then C has at least four arcs.*

Proof: By construction, D has no loops and therefore no 1-cycles, dangerous or otherwise. Similarly, 2-cycles correspond to edges of G, and therefore have two positive arcs. Finally, if $C = (a, b, c)$ were a dangerous 3-cycle (where at most arc (a, b) is positive), then it would have weight

$$\begin{aligned} w(C) &\leq \alpha_{ab} - \alpha_{bc} - \alpha_{ca} \\ &\leq 1 - 1/2 - 1/2 \\ &= 0. \end{aligned}$$

This contradicts the fact that C has positive weight. ▮

We already know by Theorem 3.7 that every strip graph is a cocomparability graph. However, Lemma 5.26 further illuminates this property, as the following alternative proof illustrates.

Theorem 3.7 (Chapter 3) Strip graphs form a subclass of cocomparability graphs.

Proof: We need to show that the complement of a strip graph can be transitively oriented. If G is a strip graph, then by Theorem 5.17 it can be levelled and complement oriented such that its corresponding digraph has only positive cycles. This orientation is transitive, for otherwise the corresponding directed graph would have a dangerous triangle, contradicting Lemma 5.26. ▮

Lemma 5.27 *Let D be the directed graph corresponding to a levelled, complement oriented strip graph G. If D has no dangerous cycles, then G is an indifference graph (a one-level graph).*

Proof: Write $D = (V, A, w)$ and define a new weighted digraph $D_0 = (V, A, w_0)$ where

$$w_0(u,v) = \begin{cases} 1 & \text{if } w(u,v) > 0 \\ -1 & \text{if } w(u,v) < 0 \\ 0 & \text{if } w(u,v) = 0 \text{ i.e., if } u = v_0 \end{cases}$$

Now consider any cycle C in D_0 (and therefore also in D). Since C is not dangerous, $p > n$. Therefore, $w_0(C) = p - n > 0$. Hence D_0 has only positive cycles. Since D_0 corresponds to G relevelled so that $y(v) = 0$ for all vertices $v \in V$, it follows that G is an indifference graph, that is, a 0-strip graph, by Corollary 5.17.1. ▮

By the previous lemma, the digraph of every proper strip graph has a dangerous cycle. This dangerous cycle must have at least four arcs by Lemma 5.26. You may therefore wonder how large the *smallest* such dangerous cycle could be. This is answered by the following lemma.

Lemma 5.28 *If the digraph corresponding to a levelled, complement oriented graph has a positive dangerous cycle, then it has a dangerous cycle with four arcs.*

Proof: Let D be the directed graph corresponding to a levelled, complement oriented graph $G = (V, E)$. Let C be a dangerous cycle in D with the least number of arcs. We may assume that C is simple, for otherwise C can be decomposed into two cycles, at least one of which is dangerous. By Lemma 5.26, C has at least four arcs.

Since C is dangerous, it has at least one negative arc. On the other hand, if C had only negative arcs, its weight would be negative. Therefore C has at least one positive arc. In particular, it has a negative arc (b, c) followed by a positive arc (c, d); see Figure 5.8. Let a precede b in C. Now, either (a, b) is positive or it is negative, and either (a, d) is an edge in G or it is not. This gives us four cases to consider:

Case 1: (a, b) is positive and $(a, d) \in E$. See Figure 5.8.

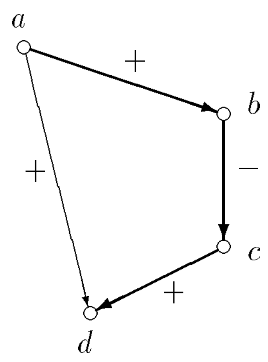


Figure 5.8: Case 1: (a, b) is positive, and (a, d) is positive.

Since (a, d) is an edge in G, it follows that (a, d) is a positive arc in D. So replacing (a, b, c, d) with (a, d) in C results in a dangerous[2] cycle with fewer arcs, contradicting the assumption that C is a shortest dangerous cycle.

Case 2: (a, b) is positive and $(a, d) \notin E$. See Figure 5.9.

Since (a, d) is not an edge of G, either (a, d) or (d, a) is a negative arc in D. If (a, d) is a negative arc in D, then replacing (a, b, c, d) with (a, d) in C results in a dangerous[3] cycle with fewer arcs, again contradicting the assumption that C is a shortest dangerous

[2] It has $n - 1$ negative arcs and $p - 1$ positive arcs.
[3] It has n negative arcs and $p - 2$ positive arcs.

cycle. On the other hand, if (d, a) is a negative arc in D, then (a, b, c, d, a) is a dangerous 4-cycle.

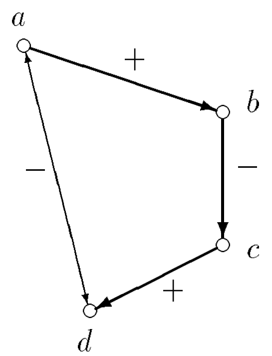


Figure 5.9: Case 2: (a, b) is positive, and (a, d) or (d, a) is negative.

Case 3: (a, b) is negative and $(a, d) \in E$. See Figure 5.10.

Since (a, d) is an edge in G, it follows that (d, a) is a positive arc in D. In this case, (a, b, c, d, a) is a dangerous 4-cycle.

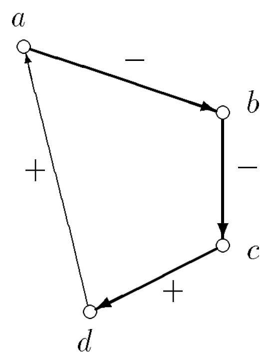


Figure 5.10: Case 3: (a, b) is negative, and (d, a) is positive.

Case 4: (a, b) is negative and $(a, d) \notin E$. See Figure 5.11.

Either (a, d) or (d, a) is a negative arc in D. If (a, d) is a negative arc in D, then replacing (a, b, c, d) with (a, d) in C results in a dangerous[4] cycle with fewer arcs, again contradicting the assumption that C is a shortest dangerous cycle. Finally, if (d, a) is a negative arc in D, then (a, b, c, d, a) is a dangerous[5] 4-cycle.

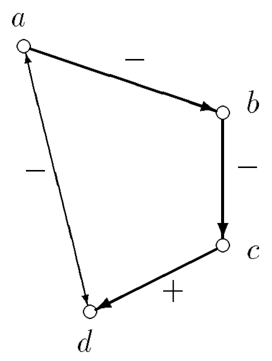


Figure 5.11: Case 4: (a, b) is negative, and (a, d) or (d, a) is negative.

▮

Lemma 5.29 *Let D be the directed graph corresponding to a levelled, complement oriented graph G. If C is a positive, dangerous 4-cycle in D, then C corresponds to a square or a claw in G.*

Proof: Let $C = (a, b, c, d, a)$ be a positive, dangerous 4-cycle. It cannot have four negative arcs, denoted $(-, -, -, -)$, since such a cycle would have negative weight. Similarly, it cannot have 3 negative arcs and one positive arc, denoted $(+, -, -, -)$, since its weight would be negative:

$$w(C) = \alpha_{ab} - \alpha_{bc} - \alpha_{cd} - \alpha_{da} \leq 1 - 3(1/2) = -1/2.$$

It follows that C has exactly two negative arcs and two positive arcs, isomorphic to either $(-, +, -, +)$ or $(-, -, +, +)$.

Suppose $C = (-, +, -, +)$ as shown in Figure 5.12. By the construction of D, the

[4]It has $n - 1$ negative arcs and $p - 1$ positive arcs.

[5]It has 3 negative arcs and 1 positive arc. Such a cycle is not only dangerous, it is unrealizable.

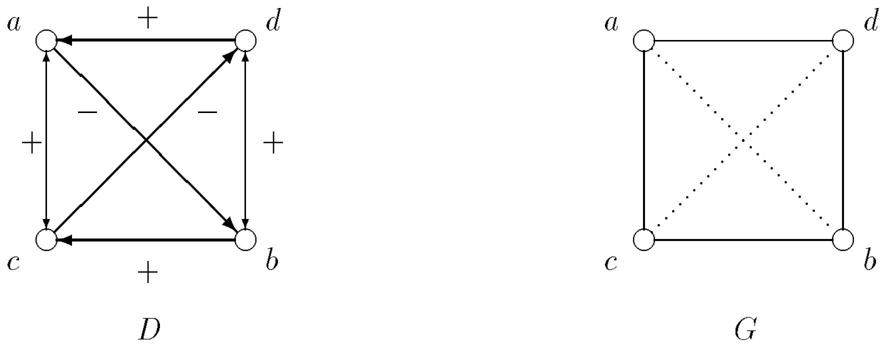


Figure 5.12: $(-,+,-,+)$ in D corresponds to a square in G.

pairs (a,d) and (b,c) are edges in G, and (a,b) and (c,d) are not. Now, arc (a,c) cannot be negative for, if it were, (a,c,d,a) would be a dangerous triangle, contradicting Lemma 5.26. Its opposite orientation, arc (c,a) also cannot be negative for, if it were, (a,b,c,a) would be a dangerous triangle. Therefore (a,c) is positive and therefore in G. Similarly, arcs (b,d) and (d,b) are not negative, since they would imply dangerous triangles (a,b,d,a) and (b,c,d,b). Therefore (b,d) is in G, and C corresponds to a square.

On the other hand, suppose $C = (-,-,+,+)$ as shown in Figure 5.13. By the

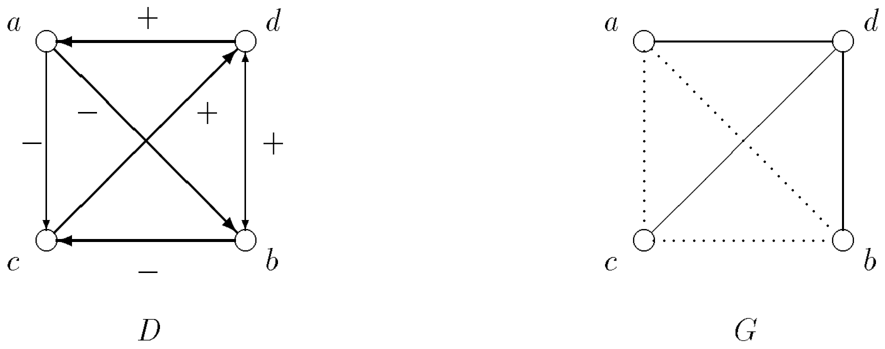


Figure 5.13: $(-,-,+,+)$ in D corresponds to a claw in G.

construction of D, the pairs (a,d) and (c,d) are edges in G, and (a,b) and (b,c) are not.

Now, edge (a, c) cannot be in G for, if it were, arc (c, a) would be positive in D, and (a, b, c, a) would be a dangerous triangle. Similarly, arcs (b, d) and (d, b) are not negative, since they would imply dangerous triangles (a, b, d, a) and (b, c, d, b). Therefore (b, d) is in G, and C corresponds to a claw. ▮

Theorem 5.30 *Any strip graph that is not also an indifference graph contains a square or a claw.*

Proof: Let G be a strip graph. It can therefore be levelled and complement oriented such that its corresponding directed graph D has only positive cycles. If G is not a one-level graph, then D has at least one dangerous cycle. By Theorem 5.28, D contains a dangerous 4-cycle, which corresponds to a square or a claw by Lemma 5.29. ▮

5.4.2 Connection with Indifference Graph Characterization

Another way of arriving at Theorem 5.30 is to examine the characterization of indifference graphs. Roberts characterized indifference graphs as claw-free interval graphs (Theorem 2.3.4). Also, Gilmore and Hoffman characterized interval graphs as chordal cocomparability graphs [GH64]. Therefore, indifference graphs are equivalently chordal, claw-free, cocomparability graphs.

We know that all strip graphs are cocomparability graphs (Theorem 3.7). Hence, if a strip graph is not an indifference graph, it must either contain a claw or not be chordal. If it is not chordal, it must contain a square, since cocomparability graphs do not have induced cycles with more than four edges (Theorem 4.6).

This higher-level proof clearly hides many of the details inherent in dangerous cycle proof of Theorem 5.30. In fact, we could prove Roberts's result (Theorem 2.3.4) using dangerous cycles. Nevertheless, the higher-level proof serves to relate strip graphs with other familiar results.

5.5 A Characterization of Trees in Strip Graphs

What sort of trees are strip graphs? If a tree has any "body" to it, that is, if it contains the subtree in Figure 5.14, then it cannot be a strip graph. The leaves of this subtree have the

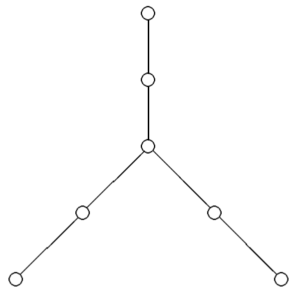


Figure 5.14: The leaves of this tree form an asteroidal triple.

property that between any pair of them, there is a path that avoids the neighbourhood of the third. A set of three vertices in a graph that has this property is called an *asteroidal triple* [LB62]. It is well known that all cocomparability graphs are asteroidal-triple free [Gal67]. This is an easy consequence of Corollary 4.7.1, that a path in a (scanning) ordered cocomparability graph dominates all vertices between its endpoints. Since all strip graphs are cocomparability graphs (Theorem 3.7), it follows that strip graphs are also asteroidal-triple free, and therefore free of the tree in Figure 5.14.

Trees that are free of this tree (Figure 5.14) are sometimes called *caterpillars*: removing their leaves results in a path (the *spine*). This definition implies that every spine vertex has degree at least 2. But for purposes of exposition, assume that the spine is ordered and that no leaves are adjacent to the first and last vertices, i.e., they have degree 1.

We already know that $K_{1,5}$ is not a strip graph (Theorem 5.19). Since strip graphs are hereditary (i.e., all induced subgraphs of a strip graph are themselves strip graphs), it follows that any tree that contains an induced $K_{1,5}$ subgraph could not be a strip graph, either.

The only trees not ruled out by this brief analysis are caterpillars all of whose vertices have degree at most 4. Remarkably, all such trees are strip graphs, as shown by the following subsection.

5.5.1 A Universal Embedding for Caterpillars

Any caterpillar with degree at most 4 is isomorphic to an induced subgraph of the infinite caterpillar shown in Figure 5.15. Any embedding of this infinite caterpillar also

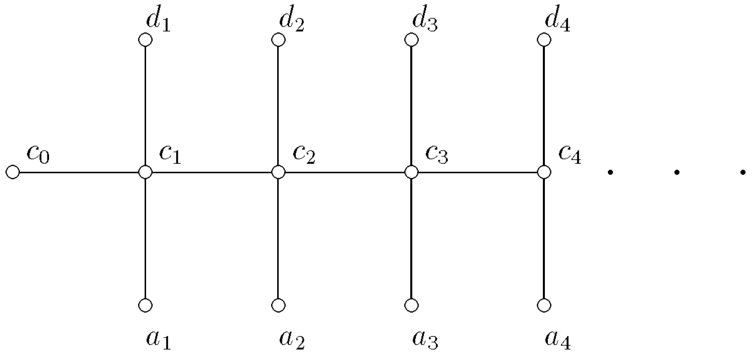


Figure 5.15: The infinite degree-4 caterpillar

specifies an embedding for an induced subgraph, and therefore for any caterpillar via its subgraph isomorphism. We will now see how to embed the infinite caterpillar. All points corresponding to vertices labelled c will lie strictly inside the strip, except for c_0 which lies on the upper level. Points corresponding to leaves labelled a will always lie on the lower level of the strip. Points corresponding to leaves labelled d will always lie on the upper level of the strip.

The following technical definition will be useful before proceeding. Say that three points a, b, and c in the strip satisfy the *iterative conditions* if

(i1) points a and b lie on the lower level,

(i2) point b is more than unit distance to the right of a, and

(i3) point c is less than unit distance from both a and b,

as shown in Figure 5.16. In any such triple of points, b will be just a construction point,

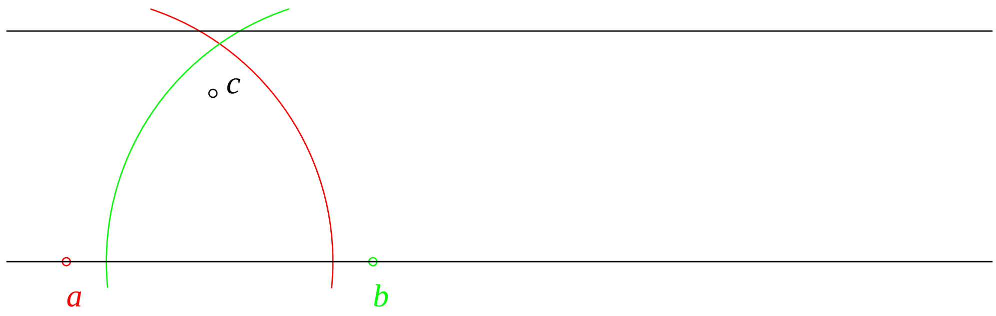


Figure 5.16: Points a, b, and c satisfy the iterative conditions.

but point a will correspond to a leaf of the infinite caterpillar labelled a, and point c will correspond to an interior vertex (of degree 4) labelled c.

The Initial Step

To embed the infinite caterpillar, we will proceed iteratively, from left to right in the strip. To this end, place c_0 on the upper level, at location $(0, \sqrt{3}/2)$. Place a_1 slightly more than unit distance from c_0, on the lower level to the right, and b_1 slightly more than unit distance from a_1, also on the lower level to the right. More concretely, place a_1 at $(0.51, 0)$ and b_1 at $(1.52, 0)$, as shown in Figure 5.17. We can then place c_1 within unit distance of c_0, a_1, and b_1. Again, more concretely, place c_1 at $(0.75, 0.5)$. Clearly, points a_1, b_1, and c_1 satisfy the iterative conditions.

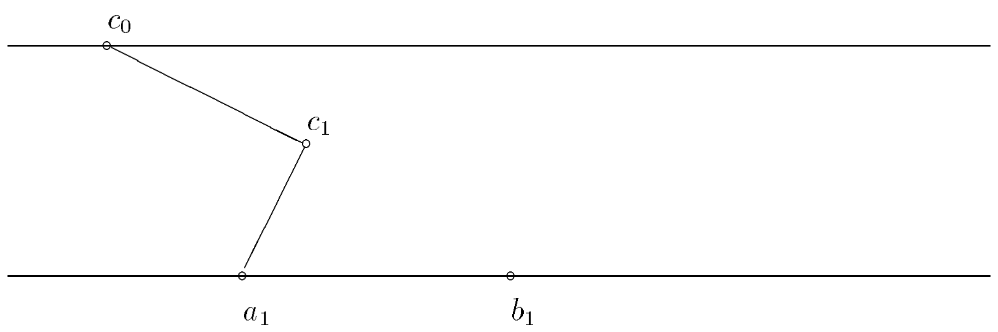


Figure 5.17: Embedding the first four vertices of the infinite degree-4 caterpillar in the strip.

The Iterative Step

We can always continue embedding the caterpillar from three points a, b, and c that satisfy the iterative conditions as follows. Place a new point d on the upper level, between a and b, exactly unit distance from b, as shown in Figure 5.18. Note that $x_d = x_b - 1/2$

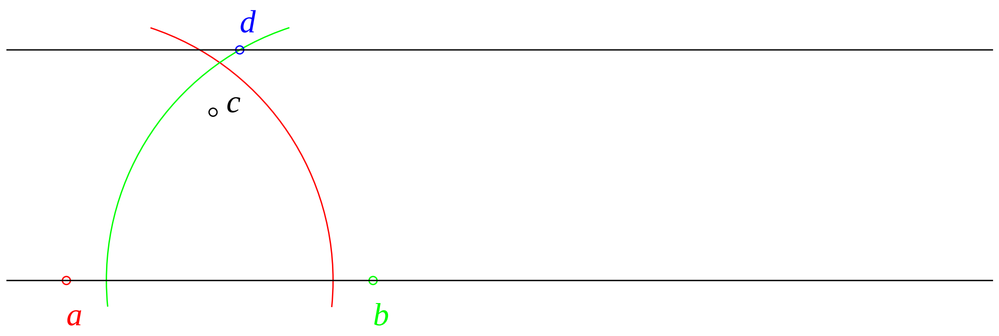


Figure 5.18: Placing point d, which corresponds to a leaf.

since the thickness of the strip is $\sqrt{3}/2$, and that $x_d > x_a + 1/2$ by condition (i2). It follows[6] that d is more than unit distance from a. Then point c is within unit distance of d. If it were not, $|x_c - x_d|$ would be greater than $1/2$, so that either $|x_c - x_a| > 1$ or $|x_c - x_b| > 1$, which contradicts the proximity of point c to points a and b. It follows that

[6]Note well that this conclusion does *not* follow if the thickness of the strip is less than $\sqrt{3}/2$. Consequently this construction is not applicable to thinner strips.

vertices a and d are independent but adjacent to vertex c in the strip graph generated by points a, c, and d.

We are now ready to construct a new set of points a', b', and d'. Let $C(p)$ denote the unit radius circle about point p, where p is any point in the strip. By construction, $C(d)$ intersects the lower level at point b, as shown in Figure 5.19.

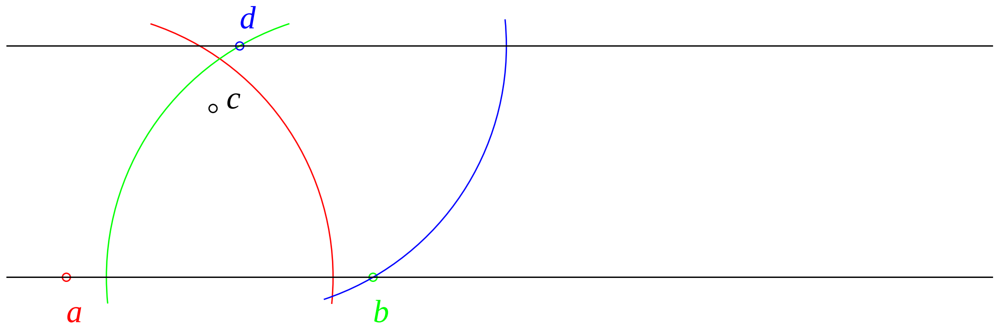


Figure 5.19: The unit circle about d intersects the lower level at point b.

By the iterative conditions, $C(c)$ intersects the lower level at a point x to the right of b, as shown in Figure 5.20. Therefore, there is an arc A of $C(c)$ that (1) lies above the lower level, (2) lies to the right of the circle about c, (3) has positive length, and (4) has a tangent with finite positive slope at all points. Figure 5.20 shows arc A in bold.

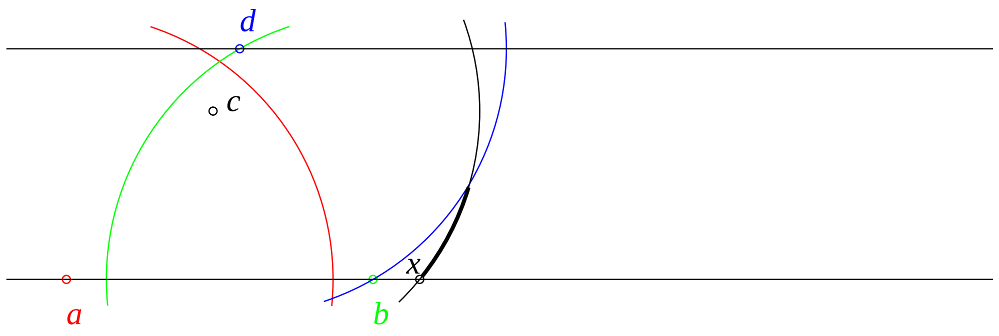


Figure 5.20: The unit circle about c intersects the lower level at point x. Arc A is the bold part of $C(c)$.

Now place b' on the lower level to the right of x such that $C(b')$ intersects arc A at its midpoint, as shown in Figure 5.21, although any point of intersection suffices.

The circle $C(b')$ intersects the lower level at a point y to the right of point x since the

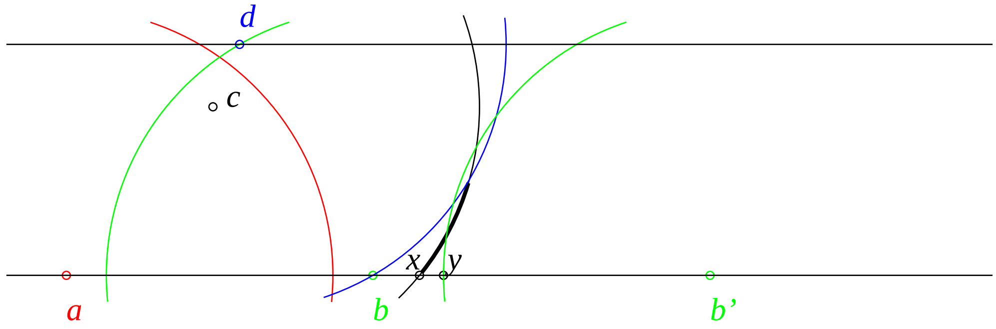


Figure 5.21: Circle $C(b')$ intersects arc A at its midpoint, and it intersects the lower level at point y.

tangent to $C(b')$ is vertical at the lower level, but the tangent to $C(c)$ has positive slope. Therefore let a' be the midpoint between x (on the left) and y (on the right), as shown in Figure 5.22, although any point between x and y will suffice. This ensures that a' and b' satisfy conditions (i1) and (i2).

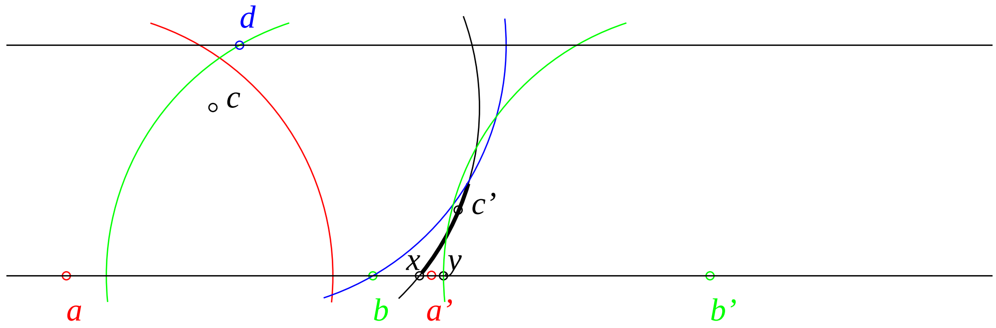


Figure 5.22: Point c' is on $C(c)$ and inside $C(b')$

Finally, let c' be the point on A midway between the top of A and the intersection of A and $C(b')$, as shown in Figure 5.22, though any point on A that lies inside $C(b')$ will suffice. Since c' is inside $C(b')$, it is within unit distance of b'. We need to show that c' is also within unit distance of a'. Since any continuous intersection of a circle (centered in the strip) with the strip (of width $\sqrt{3}/2$) has diameter at most 1, it follows that arc A has diameter at most 1. As a consequence, c' is within unit distance of x. Therefore c'

is within unit distance of a', which lies between x and b', by convexity of the unit-radius disk about c'.

In summary, a', b', and c' satisfy the iterative conditions. Furthermore, a' is not adjacent to a or d since it lies to the right of b, and it is not adjacent to c since it lies to the right of x. Similarly, c' is not adjacent to a or d since it lies outside of $C(d)$. Figure 5.23 shows that portion of the caterpillar generated by points $\{a, c, d, a', c', d'\}$,

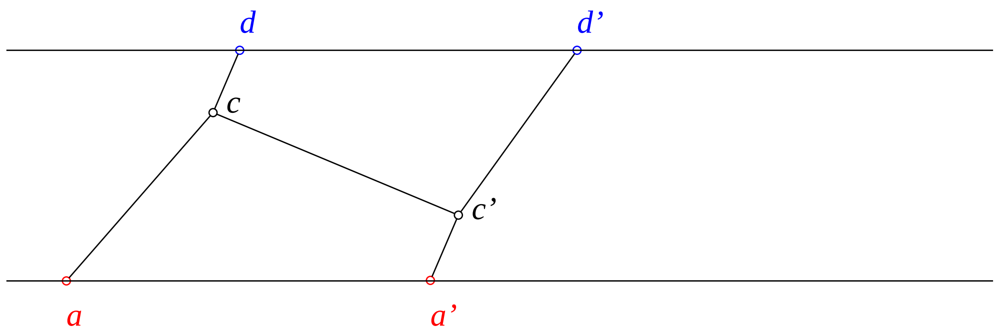


Figure 5.23: The strip graph generated by $\{a, c, d, a', c', d'\}$.

where d' is unit distance from b'. Note that d' is not adjacent to a, b, or c. To see this, recall that points a, b and c have x-coordinates less than $x_{a'} - 1/2$, and that d' has an x-coordinate greater than $x_{a'} + 1/2$. It follows that a, b and c have x-coordinates less than $x_{d'} - 1$ and therefore lie more than unit distance from d'. We have established the following theorem.

Theorem 5.31 *A tree is a $\sqrt{3}/2$-strip graph if and only if it is a caterpillar with degree at most 4.*

Chapter 6

Two-Level Graphs

This chapter studies two-level strip graphs, that is, graphs that have strip-realizations with only two distinct y-coordinates. Since the y-coordinates of arbitrary strip graphs can take on an infinite number of values, it is difficult to see the combinatorial aspects of the y-dimension. To emphasize these combinatorial aspects, we define a notion of k-level graphs, in which the y-coordinates take on at most k distinct values. If $k = 1$, these one-level graphs are exactly the indifference graphs. For $k = 2$, the main object of attention in this chapter, each of the two "levels" can be thought of as an indifference graph. But still, two-level graphs are *not* indifference graphs. It therefore behooves us to to better understand the relation between indifference graphs and two-level graphs. We will make significant steps towards characterizing, recognizing, and laying out (*realizing*) two-level graphs. We will also see how to develop algorithms for this class of graphs.

After a few definitions, we will look at some basic examples in Section 6.2. In particular, we will see that both the square and the claw are two-level graphs (although neither are indifference graphs), but that their realizations are highly constrained. This immediately applies to show that $K_{1,4}$, and consequently any star with more than three leaves, is not a two-level graph. Finally, we will look at a small graph whose realization is completely constrained, that is, whose levels and complement orientation is completely determined. This small graph will be useful for designing larger graphs in which we wish to restrict the choice for the level of a vertex.

Next, we will see that the geometric realization of k-level graphs can be used to develop efficient algorithms. Section 6.3 illustrates this fact by developing an efficient algorithm for solving a dominating set problem. This problem, finding a minimum weight independent dominating set, was solved (less efficiently) in the more general context of cocomparability graphs by Chapter 4. The problem is equivalent to finding a minimum weight maximal clique in the complement of a two-level graph; this formulation yields the necessary insights for solving the problem.

Section 6.4 examines the relation of two-level graphs with other classes of graphs. In particular, it places two-level graphs farther down a small hierarchy of previously studied perfect graphs that contains cocomparability graphs. We will see that a two-level graph under the Euclidean metric is also a two-level graph under the L_1 "city block" metric. We will also see how to compute two illustrative parameters of two-level graphs, by showing that both the interval number and the boxicity of two-level graphs is at most two. Finally, we will see that the edges of a two-level graph can be bipartitioned such that one set induces an indifference graph and the other induces a bipartite permutation graph.

The last section, Section 6.5, examines the problem of recognizing two-level graphs. We will begin by assuming that we know which level every vertex must be on (i.e., the graphs are *striated*), even though we might not know the y-coordinate of that level. We will see that the complements of striated two-level graphs are uniquely orientable. Striated k-level graphs, for $k > 2$, are not in general uniquely complement orientable, nor are two-level graphs that have not been striated. However, we will see that any transitive orientation of the complement of an unstriated two-level graph is compatible with some realization. Unfortunately, we will also see that, unlike indifference and cocomparability graphs, there is no forbidden ordered-triple characterization for two-level graphs. We will examine the structure of the class of two-level graphs as the thickness of the strip varies. In particular, we will see that the classes for any two distinct thicknesses are incomparable. Nevertheless, if a graph is both a two-level τ_1-strip graph, and a two-level τ_2-strip graph, then it is a two-level τ-strip graph for all values of τ between τ_1 and τ_2. The section concludes by showing how to recognize striated two-level graphs in $O(V^3 \log V)$ time with the mechanism developed for levelled, complement oriented strip graphs.

6.1 Definitions

Let $G = (V, E)$ be a τ-strip graph, and let $f : V \to \mathbf{R} \times [0, \tau]$ be a realization of G. As usual, we write $f(u) = (x_f(u), y_f(u))$, and $\alpha_{uv} = \sqrt{1 - (y_f(u) - y_f(v))^2}$.

Definition 6.1 A τ-strip graph $G = (V, E)$ is a *k-level (τ-strip) graph* if there is a set $\{y_1, y_2, \ldots, y_k\}$ of real values where $0 = y_1 < y_2 < \cdots < y_k = \tau$, such that G has a realization f satisfying $y_f(u) \in \{y_1, y_2, \ldots, y_k\}$ for every vertex $u \in V$. Say that *vertex v is on level i*, or *on the ith level*, if $y_f(v) = y_i$. ∎

Note that $\alpha_{uv} = \alpha_{uw}$ whenever v and w are on the same level. Sometimes we would like to require the vertices to be on certain levels, without actually saying what the y-values of those levels are. To this end, let us define a notion of striation, which captures this intuition.

Definition 6.2 A *k-striation* is a function $s : V \to \{1, 2, \ldots, k\}$. A graph $G = (V, E, s)$ is *k-striated* if it includes a k-striation function s. A striated graph $G = (V, E, s)$ is a *striated k-level (τ-strip) graph* if there is a realization $f : V \to \mathbf{R}^2$ and a set of k y-values (*striae*) $\{y_1, y_2, \ldots, y_k\}$ such that $0 = y_1 < y_2 < \cdots < y_k = \tau$, and $y_f(u) = y_{s(u)}$, for every vertex $u \in V$. ∎

Definition 6.3 If the endpoints of an edge in a striated graph (or levelled graph) are on the same stria (or level), call it a *level edge*, otherwise call it a *cross edge*. ∎

One-level graphs are of course (one-dimensional) indifference graphs; there is no harm in assuming that the unused y-coordinate of all vertices is 0, and that all edges are level edges. Two-level graphs, for which $k = 2$, are clearly included in these definitions. For example, if f is a two-level realization for $G = (V, E)$, then vertex $v \in V$ is on level 1 if $y_f(v) = 0$, and on level 2 if $y_f(v) = \tau$. In addition, since two-level graphs are discussed extensively in this chapter, it is convenient to have the following definition. The intention behind this definition is given by Property 6.5 below.

Definition 6.4 Let $\alpha = \sqrt{1-\tau^2}$ be the *critical x-dimension* corresponding to the strip thickness τ. Note that, if $0 \leq \tau \leq \sqrt{3}/2$, then $1/2 \leq \alpha \leq 1$. Note also that, if the variable τ is symbolic (its value has not been specified), then so is α. ▮

Property 6.5 *Let f be a two-level realization for G. For all vertices u and v on the same level, $(u, v) \in E$ if and only if $|x_f(u) - x_f(v)| \leq 1$. For all u and v on different levels, $(u, v) \in E$ if and only if $|x_f(u) - x_f(v)| \leq \alpha$. In either case, $(u, v) \in E$ if $|x_f(u) - x_f(v)| \leq \alpha$.*

Proof: If u and v are on the same level, then $y_f(u) - y_f(v) = 0$. It follows that

$$\begin{aligned}
(u,v) \in E \quad & \text{if and only if} \quad \|f(u) - f(v)\| \leq 1 \\
& \text{if and only if} \quad \sqrt{(x_f(u) - x_f(v))^2 + (y_f(u) - y_f(v))^2} \leq 1 \\
& \text{if and only if} \quad \sqrt{(x_f(u) - x_f(v))^2} \leq 1 \\
& \text{if and only if} \quad |x_f(u) - x_f(v)| \leq 1.
\end{aligned}$$

Similarly, if u and v are on different levels, then $|y_f(u) - y_f(v)| = \tau$. It follows that

$$\begin{aligned}
(u,v) \in E \quad & \text{if and only if} \quad \|f(u) - f(v)\| \leq 1 \\
& \text{if and only if} \quad \sqrt{(x_f(u) - x_f(v))^2 + (y_f(u) - y_f(v))^2} \leq 1 \\
& \text{if and only if} \quad \sqrt{(x_f(u) - x_f(v))^2 + \tau^2} \leq 1
\end{aligned}$$

$$\begin{aligned}
&\text{if and only if} \quad (x_f(u) - x_f(v))^2 + \tau^2 \leq 1 \\
&\text{if and only if} \quad |x_f(u) - x_f(v)| \leq \sqrt{1-\tau^2} \\
&\text{if and only if} \quad |x_f(u) - x_f(v)| \leq \alpha.
\end{aligned}$$

In either case $(u, v) \in E$ if $|x_f(u) - x_f(v)| \leq \alpha$ since $\alpha \leq 1$. ∎

6.2 Examples

Consider some realization of a two-level graph $G = (V, E)$. Any cycle in the complete graph $K(V)$ on V (or any graph on V, for that matter) must have an even number of cross edges. This is because any cycle must begin and end on the same level. In particular, this is true of any directed cycle in the weighted digraph (Definition 5.16) corresponding to G, the left-to-right orientation, and this levelling. Cross arcs in the digraph have weight $\pm\alpha$, and level arcs have weight ± 1 by Property 6.5. Recall that $\alpha = \sqrt{1-\tau^2}$ and that $1/2 \leq \alpha \leq 1$. Furthermore, arcs in the digraph that correspond to edges in the graph have positive weight, and arcs that correspond to nonedges have negative weight. The next two lemmas rely on Corollary 5.17.1, that a graph is a τ-strip graph if and only if it can be τ-levelled and complement oriented such that every cycle in its corresponding weighted digraph has positive weight.

Lemma 6.6 *Let (a, b, c, d, a) be a chordless 4-cycle (a square) in a two-level graph $G = (V, E)$. Then (a, d) and (b, c) are level edges, and the others are cross edges, for every two-level realization f of G for which $x_f(a) < x_f(c)$ and $x_f(b) < x_f(d)$. That is, the orientation of a square determines its levels.*

Proof: Let the vertex set $\{a, b, c, d\}$ induce a square in a two-level graph G as shown in Figure 6.1.(i). Let f be a two-level realization for which $x_f(a) < x_f(c)$ and $x_f(b) <$

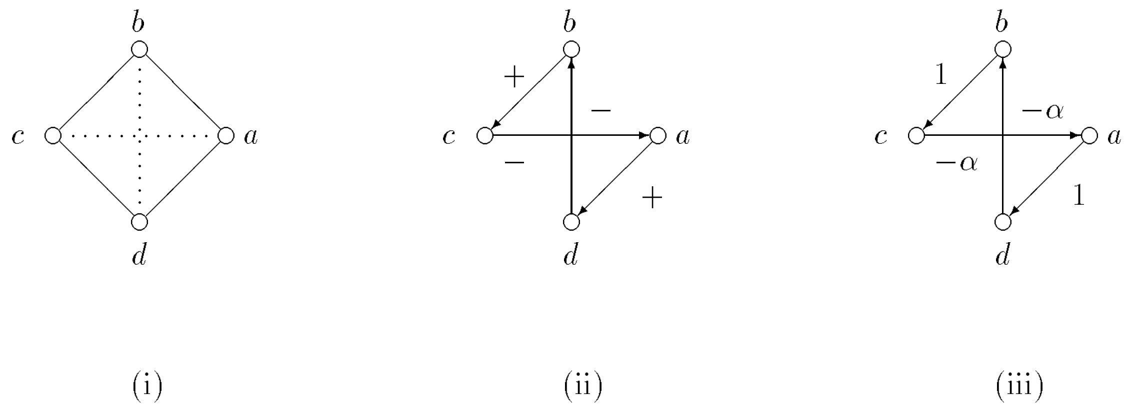


Figure 6.1: An induced square in a complement oriented two-level graph, and a cycle in the corresponding weighted digraph.

$x_f(d)$. Then (a, d, b, c, a) is a cycle in the weighted digraph corresponding to graph G and realization f. Arcs (a, d) and (b, c) correspond to edges in G and so have positive weight, while arcs (d, b) and (c, a) have negative weight, as shown in Figure 6.1.(ii). The cycle must have an even number of cross arcs. However, it cannot have 0 or 4 cross arcs, since its weight would then be 0. Therefore it has exactly two cross arcs and two level arcs. The level arcs must be the positive arcs (a, d) and (b, c) as shown in Figure 6.1.(iii), since otherwise the cycle would have nonpositive weight. ∎

Lemma 6.7 *Let $G(\{a, b, c, d\})$ be an induced claw, where b is the "hub", in a two-level graph $G = (V, E)$. Then (a, b) and (b, c) are level edges, and (b, d) is a cross edge, for every two-level realization f of G for which $x_f(a) < x_f(d)$ and $x_f(d) < x_f(c)$. That is, the orientation of a claw determines its levels.*

Proof: Let the vertex set $\{a, b, c, d\}$ induce a claw in a two-level graph G as shown in Figure 6.2.(i). Let f be a two-level realization for which $x_f(a) < x_f(d)$ and $x_f(d) < x_f(c)$. Then (a, b, c, d, a) is a cycle in the weighted digraph corresponding to graph G and realization f. Arcs (a, b) and (b, c) correspond to edges in G and so have positive weight,

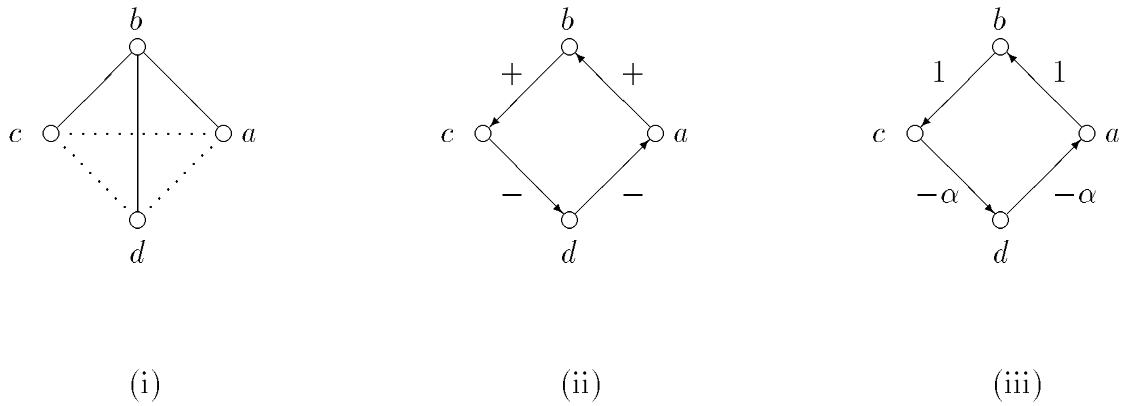


Figure 6.2: An induced claw in a complement oriented two-level graph, and a cycle in the corresponding weighted digraph.

while arcs (c, d) and (d, a) have negative weight, as shown in Figure 6.2.(ii). The cycle must again have an even number of cross arcs, and again it cannot have 0 or 4 cross arcs, since its weight would then be 0. Therefore it has exactly two cross arcs and two level arcs. The level arcs must be the positive arcs (a, b) and (b, c) as shown in Figure 6.2.(iii), since otherwise the cycle would have nonpositive weight. ∎

Lemma 6.8 *$K_{1,4}$ is not a two-level graph.*

Proof: Suppose there is a two-level realization for $K_{1,4}$. Either there is an induced claw with all vertices on the same level, or there is an induced claw with two leaf vertices on a level different from the hub's level. Neither case is allowed by Lemma 6.7. ∎

Lemma 6.9 *Let $\{a, b, c, d, e, f\}$ induce the subgraph G' in Figure 6.3 in some two-level graph. Then any two-level realization orients and levels G' as shown in the figure (orient all complement edges left-to-right).*

Proof: The subgraph G' is clearly a two-level graph by the realization shown in the figure. It is therefore a cocomparability graph. In fact, its complement is uniquely

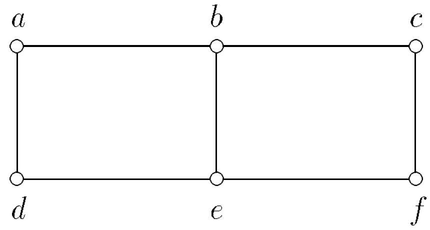


Figure 6.3: A uniquely levelled, uniquely complement oriented, two-level graph.

orientable. To see this, start with nonedge (a, f). Recall from Definition 4.49 that an arc (a, b) *directly forces* arc (c, d) (that is, $(a, b)\Gamma(c, d)$) if

1. $a = c$ and $(b, d) \notin \overline{E}$, or

2. $b = d$ and $(a, c) \notin \overline{E}$.

The following relations are immediate.

$$\begin{array}{lllll}
(a, f) & \Gamma & (a, c) & \Gamma & (d, c) \\
(a, f) & \Gamma & (a, e) & & \\
(a, f) & \Gamma & (b, f) & & \\
(a, f) & \Gamma & (d, f) & & \\
(d, c) & \Gamma & (e, c) & & \\
(d, c) & \Gamma & (d, b) & &
\end{array}$$

Therefore (a, f) forces the seven remaining nonedges. For example,

$$(a, f)\Gamma(a, c)\Gamma(d, c)\Gamma(e, c).$$

Consequently, if $x(a) < x(f)$ for some realization, then

$$\begin{array}{lllll}
x(a) & < & x(e) & < & x(c), \\
x(a) & < & x(f), & & \\
x(d) & < & x(b) & < & x(f), \\
x(d) & < & x(c). & &
\end{array}$$

Now since (a, b, d, e, a) and (b, c, e, f, b) are chordless 4-cycles, two applications of Lemma 6.6 show that (a, b), (d, e), (b, c), and (e, f) are level edges, and all others are cross edges. ▮

6.3 Exploiting a Geometric Realization

This section develops a data structure that implicitly represents the oriented complement of a k-level graph $G = (V, E, \vec{E})$ given its realization $f : V \to \mathbf{R}^2$. We will see that this data structure also efficiently represents the transitive reduction (Definition 4.32) of the oriented complement $\vec{G} = (V, \vec{E})$. In fact, the forthcoming Theorem 6.22 proves that the transitive reduction (V, R) of $\vec{G}$ can be computed in $O(V \log V + k^2 V + R)$ time. This compares with $O(M(V))$, the best known time for the transitive reduction of arbitrary digraphs [AGU72], and $O(VR)$, the best known time for transitively oriented comparability graphs (§4.2.4).

This section applies the data structure to solve the weighted independent dominating set problem on k-level graphs in $O(k^2 V + V^2)$ time. In particular, it solves this problem on two-level graphs in quadratic time. This problem is equivalent to the weighted maximal clique problem on the complements of k-level graphs. For fixed k, these algorithms therefore improve on the $O(M(V))$ time algorithms for cocomparability and comparability graphs in Chapter 4.

6.3.1 k-Level Graphs and their Oriented Complements

Definition 6.10 A digraph $\vec{G} = (V, \vec{E})$ is the *oriented complement (of strip graph $G = (V, E)$ and realization f)* if $\vec{E} = \{(u, v) : (u, v) \in \overline{E} \text{ and } x_f(u) < x_f(v)\}$. ▮

The next property follows immediately from this definition and that of α_{uv} (Definition 5.4).

Property 6.11 *Let $\vec{G} = (V, \vec{E})$ be the oriented complement of a k-level graph G and realization f. The ordered pair (u, v) is an arc in $\vec{E}$ if and only if $x_f(v) - x_f(u) > \alpha_{uv}$.*

Another immediate corollary is due to k-level graphs also being strip graphs.

Property 6.12 *If $\vec{G} = (V, \vec{E})$ is the oriented complement of k-level graph $G = (V, E)$ and realization f, then $\vec{E}$ is a transitive orientation of $\overline{E}$.*

6.3.2 An Implicit Representation of the Oriented Complement

This subsection shows how to create a representation of the oriented complement $\vec{G} = (V, \vec{E})$ given its realization f. The forthcoming algorithms are easier to express if we augment V with two sentinel vertices s_i and t_i on each level i. To this end, let

$$V' = V + \{s_i : 1 \leq i \leq k\} + \{t_i : 1 \leq i \leq k\}.$$

Note that $|V'| = |V| + 2k = O(V)$ since $k \leq V$. Ensure that each s_i is adjacent to every other vertex in $\vec{G}$ by locating it more than unit distance to the left of the leftmost vertex in V. That is, set $x_f(s_i)$ such that $x_f(s_i) < \min\{x_f(u) : u \in V\} - 1$ for all i. Similarly, ensure that each t_i is adjacent to every other vertex in $\vec{G}$ by locating it more than unit distance to the right of the rightmost vertex in V. That is, set $x_f(t_i)$ such that $x_f(t_i) > \max\{x_f(u) : u \in V\} + 1$ for all i.

Assume that V' is *ordered by level*, that is, $u \preceq v$ if and only if $x_f(u) < x_f(v)$, or $x_f(u) = x_f(v)$ and $y_f(u) \leq y_f(v)$, for all vertices $u, v \in V'$. The following observation is an immediate consequence of this assumption.

Observation 6.13 *Let $u, v \in V'$ be two vertices on the same level. If $u \preceq v$, then $x(u) \leq x(v)$.*

Definition 6.14 Let V' be a set of vertices ordered by level. Let L, p, and n be arrays, where L_v denotes the level of vertex v, p_i denotes the first (the premier) vertex of level i, and n_i denotes the number of vertices on level i. ∎

The following straightforward algorithm computes these values in a single pass over V', once ordered. Notice that $s_i = p_i$ and that $t_i = p_i + n_i + 1$. Notice also that $\{p_i+1, p_i+2, \ldots, p_i+n_i\}$ are the vertices from V on level i, and that $\sum_{i=1}^{k} n_i = 2k+V = O(V)$.

Table 6.1: **Algorithm:** INITIALIZE-ORIENTED-COMPLEMENT(V, f)

Input: The vertices V and a realization f of a k-level graph $G = (V, E)$.
Output: Arrays L_v, p_i, and n_i (Definition 6.14)

1 $V' \leftarrow V \cup \{s_i : 1 \le i \le k\} \cup \{t_i : 1 \le i \le k\}$
2 Order (relabel) V' by level so that $V' = \{1, 2, \ldots, |V'|\}$.
3 $i \leftarrow 0$ ▷ i is the current level
4 $y_0 \leftarrow -1$ ▷ Another sentinel
5 **for** $v \leftarrow 1$ to $|V'|$
6   **do if** $y_f(v) > y_i$ ▷ Check if v is on a new level
7   **then** $i \leftarrow i+1$
8             $n_i \leftarrow 0$
9             $p_i \leftarrow v$
10       $L_v \leftarrow i$
11       $n_i \leftarrow n_i + 1$

12 **return** (L, p, n)

Lemma 6.15 *Algorithm* INITIALIZE-ORIENTED-COMPLEMENT(V, f) *computes the arrays L, p, and n in $O(V \log V)$ time.*

Proof: Correctness follows from Definition 6.14. Since the realization function y is onto, Step 9 sets p_i every time Step 7 changes the level.

Step 2 is time critical, and takes $O(V' \log V') = O(V \log V)$ time. Steps 7, 8, and 9 are each executed k times. The other atomic steps in the loop are executed once for each vertex in V'. ▮

Definition 6.16 For every vertex $u \in V$ and level $1 \le i \le k$, define $F_i[u]$ to be the least vertex on level i for which $(u, F_i[u]) \in \vec{E}$. ▮

Figure 6.4 illustrates this notion for two levels of a graph. The presence of vertex t_i

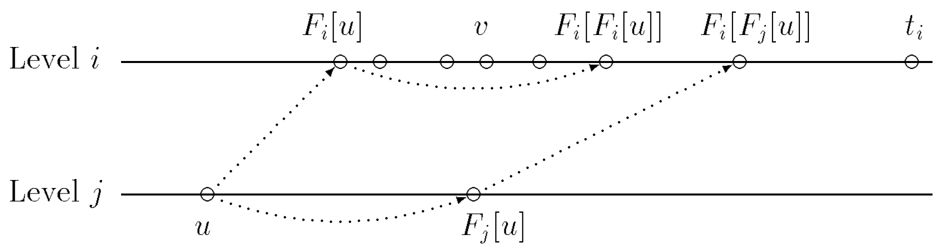


Figure 6.4: Definition of array F: value $F_i[u]$ points to the first vertex on level i that is adjacent to u in the oriented complement graph.

ensures that $F_i[u]$ is always well defined. We can implement array F as a one-dimensional array containing k one-dimensional arrays of size n_i each, where $1 \le i \le k$. Recall that $\sum_{i=1}^{k} n_i = 2k + V$ so that $|F| = O(k + (2k + V)) = O(V)$. The arrays F and L constitute an implicit representation of ordered complement $\vec{G}$ with $O(V)$ size, as illustrated by the following simple observation.

Lemma 6.17 *Let $\vec{G} = (V, \vec{E})$ be an oriented complement graph. The ordered pair (u, v) is an arc in $\vec{E}$ if and only if $F_i[u] \preceq v$, where $i = L_v$.*

Proof: If $(u, v) \in \vec{E}$, then $F_i[u] \preceq v$ by Definition 6.16.

Conversely, if $F_i[u] \preceq v$, then $x(F_i[u]) \le x(v)$ by Observation 6.13. Furthermore $(u, F_i[u]) \in \vec{E}$ by Definition 6.16, so $x(u) + \alpha_{u,F_i[u]} < x(F_i[u])$ by Property 6.11. Finally

$\alpha_{uv} = \alpha_{u,F_i[u]}$ (see Definition 5.4) since v and $F_i[u]$ are on the same level. It follows that

$$\begin{aligned} x(u) + \alpha_{uv} &= x(u) + \alpha_{u,F_i[u]} \\ &< x(F_i[u]) \\ &\leq x(v). \end{aligned}$$

Therefore $(u, v) \in \vec{E}$, also by Property 6.11. ∎

The remaining algorithms in this section do not make explicit use of L_v, since they know the level of every vertex. We conclude by showing that inequalities are preserved by "taking F of both sides", if both sides are on the same level.

Lemma 6.18 *Let $G = (V, \vec{E})$ be an oriented complement graph. For all levels i, if $u \preceq v$ and u and v are on the same level, then $F_i[u] \preceq F_i[v]$.*

Proof: Suppose u and v are on the same level j and that $u \preceq v$. Then $x(u) \leq x(v)$ by Observation 6.13. Now consider any level i. Since $(v, F_i[v]) \in \vec{E}$ and u and v are on the same level, Property 6.11 implies that $x(F_i[v]) - x(v) > \alpha_{v,F_i[v]} = \alpha_{u,F_i[v]}$. Combining these inequalities

$$x(F_i[v]) - x(u) \geq x(F_i[v]) - x(v) > \alpha_{u,F_i[v]}$$

implies that $(u, F_i[v]) \in \vec{E}$ by Property 6.11. Finally, $F_i[u] \preceq F_i[v]$ by Lemma 6.17. ∎

6.3.3 An Algorithm for Constructing the Representation

We are now ready to describe an algorithm for constructing the array F (Definition 6.16). We will begin by describing an algorithm that, given levels i and j, computes $F_i[u]$ for every vertex u on level j.

Theorem 6.19 *Given an oriented complement, represented by array p (Definition 6.14) and realization f, of a k-level strip graph, Algorithm TWO-LEVEL-F generates the ith row of array F (Definition 6.16) in $O(n_i + n_j)$ time.*

Table 6.2: **Algorithm:** TWO-LEVEL-F(i, j, p, n, f)

Input: Level numbers i and j,
arrays p and n (Definition 6.14), and
realization f of a k-level graph.
Output: The row F_i of array F (Definition 6.16).

1 $v \leftarrow p_i + 1$ ▷ Recall $s_i = p_i$ has no incoming arcs in $\vec{G}$
2 $F_i[p_j] \leftarrow v$ ▷ Recall $s_j = p_j$ is adjacent to all vertices in $\vec{G}$
3 **for** $u \leftarrow p_j + 1$ to $p_j + n_j$ ▷ u is on level j
4   **do while** $\|f(u) - f(v)\| \leq 1$
5   **do** $v \leftarrow v + 1$
6     $F_i[u] \leftarrow v$

7 **return** F_i

Proof: The algorithm computes $F_i[u]$ for vertices u on level j. It does so by simultaneously scanning the jth level with vertex u, and the ith level with vertex v, which is a candidate for $F_i[u]$. Step 1 initiates the process of scanning v along level i by pointing v to the first "real" vertex on level i. Step 3 steps through each vertex u on level j. The algorithm maintains the invariant that $v \preceq F_i[u]$, in particular at Step 3; this follows from Lemma 6.18. Steps 4 and 5 then scan v to the right until an arc $(u, v) \in \vec{E}$ is discovered. By the invariant, v is the first vertex on level i that is adjacent from vertex u, as required.

To compute the run time, note that u is set n_j times by Step 3, and v is set at most n_i times by Step 5 (remember to count the sentinel t_i) for a total of $n_i + n_j$ operations. More formally, the expression below is the number of times the algorithm executes Step 5; the sums correspond to Lines 3 and 4. We simply note that Step 5 is called $F_i[u] - F_i[u-1]$

times by Step 4, and that $p_i + 1 \preceq F_i[u] \preceq t_i = p_i + n_i + 1$.

$$\begin{aligned}
\sum_{u=p_j+1}^{p_j+n_j} \sum_{v=F_i[u-1]+1}^{F_i[u]} 1 &= \sum_{u=p_j+1}^{p_j+n_j} F_i[u] - F_i[u-1] \\
&= F_i[p_j + n_j] - F_i[p_j] \\
&= p_i + n_i + 1 - (p_i + 1) \\
&= n_i
\end{aligned}$$

▮

We are now ready to compute all of F. The algorithm below simply calls algorithm TWO-LEVEL-F as a subroutine for every pair of levels i and j.

Table 6.3: **Algorithm:** GENERATE-F(V, f)

Input: Array V of vertices, and k-level realization f of a k-level graph.
Output: The array F (Definition 6.16).

1 $\{L, p, n\} \leftarrow$ INITIALIZE-ORIENTED-COMPLEMENT(V, f)
2 **for** $j \leftarrow 1$ to k
3   **do for** $i \leftarrow 1$ to k
4         $F_i \leftarrow$ TWO-LEVEL-F(i, j, p, n, f)

5 **return** F

Theorem 6.20 *Given an oriented complement of a k-level strip graph represented by a realization f, Algorithm GENERATE-F generates array F (Definition 6.16) in $O(V \log V + kV)$ time.*

Proof: Correctness follows from Lemma 6.15 and Theorem 6.19. Step 1 takes $O(V \log V)$ time by Lemma 6.15. By Theorem 6.19, Step 4 takes $O(n_i + n_j)$ time. The sum of the number of vertices on each level over all levels is just the total number of vertices. This observation yields the following expression for the total time due to Step 4.

$$\begin{aligned}
\sum_{j=1}^{k}\sum_{i=1}^{k} O(n_i + n_j) &= O(\sum_{j=1}^{k}\sum_{i=1}^{k} n_i) \\
&= O(\sum_{j=1}^{k} V) \\
&= O(kV).
\end{aligned}$$

▮

Corollary 6.20.1 *Given an oriented complement of a two-level graph represented by a realization f, Algorithm GENERATE-F generates array F (Definition 6.16) in $O(V \log V)$ time.*

6.3.4 An Implicit Representation of the Transitive Reduction

Interestingly, array F is also an implicit representation of the transitive reduction of the oriented complement. With the aid of Lemma 6.21 below, Algorithm kTRANS (Table 6.4) generates, for any constant number k of levels, all arcs of the transitive reduction in constant time each, given the array F.

Lemma 6.21 *Let $\vec{G} = (V, \vec{E})$ be the oriented complement of a k-level graph, and let w be a vertex on level i. The ordered pair (u, w) is an arc in the transitive reduction of $\vec{G}$ if and only if $F_i[u] \preceq w \prec F_i[F_j[u]]$ for all levels j.*

Proof: Suppose first that (u, w) is an arc in the transitive reduction of $\vec{G}$. Then $(u, w) \in \vec{E}$, so that $F_i[u] \preceq w$ by Lemma 6.17. Also, if $F_i[F_j[u]] \preceq w$ for any level j, then $(F_j[u], w) \in \vec{E}$ by Lemma 6.17, which together with $(u, F_j[u])$, would transitively imply (u, w). Therefore $w \prec F_i[F_j[u]]$.

Conversely, suppose that (u, w) is not an arc in the transitive reduction of $\vec{G}$. Then either $(u, w) \notin \vec{E}$ or (u, w) is transitively implied. If $(u, w) \notin \vec{E}$, then $w \prec F_i[u]$ by

Lemma 6.17. On the other hand, suppose (u, w) is transitively implied by some chain of arcs $(u, v), (v, v_1), (v_1, v_2), \ldots, (v_k, w)$ in $\vec{E}$. Then $(v, w) \in \vec{E}$ since $\vec{G}$ is a *transitive* orientation of $\overline{G}$. Therefore (u, w) is transitively implied by $(u, v) \in \vec{E}$ and $(v, w) \in \vec{E}$. Then $F_i[v] \preceq w$ by Lemma 6.17. If $j = L_v$, as shown in Figure 6.5, then $F_j[u] \preceq v$ by

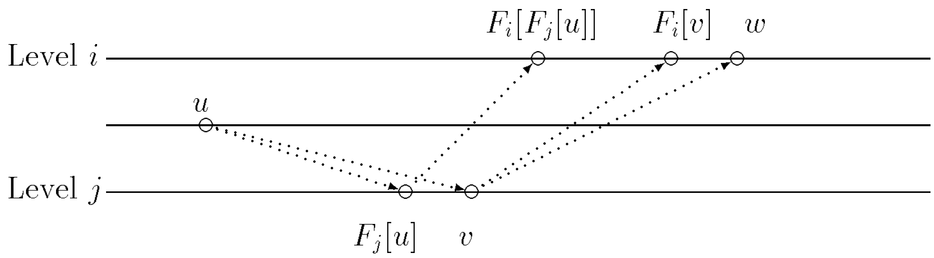


Figure 6.5: Arc (u, w) is transitively implied by (u, v) and (v, w).

Lemma 6.17, and taking F of both sides, $F_i[F_j[u]] \preceq F_i[v]$ by Lemma 6.18. Therefore $F_i[F_j[u]] \preceq F_i[v] \preceq w$. ∎

Table 6.4: **Algorithm:** kTRANS(V, f)

Input: The vertices V and a realization f of a k-level graph $G = (V, E)$.
Output: The arcs in the transitive reduction of $\vec{G}$, where $\vec{G}$ is the orientation of $\overline{G}$ that is compatible with f.

$F \leftarrow$ GENERATE-F(V, f)
for $u \in V$
  do for $i \leftarrow 1$ to k
    do $r_{iu} \leftarrow \min\{F_i[F_j[u]] : 1 \leq j \leq k\}$
      for $v \leftarrow F_i[u]$ to $r_{iu} - 1$
        do return (u, v)

Theorem 6.22 *Given a k-level strip graph, represented by vertices V and realization f, Algorithm kTRANS generates all arcs in the transitive reduction (V, R) of the graph's oriented complement in $O(V \log V + k^2 V + R)$ time.*

Proof: Correctness follows from Lemma 6.21 and Theorem 6.20. In particular, Step 6 is executed $|R|$ times.

Step 1 takes $O(V \log V + kV)$ by Theorem 6.20. Computing the minimum value of $F_i[F_j[u]]$ in Step 4 takes k operations. Since this step is called $|V|k$ times, this accounts for $O(k^2 V)$ operations.

Note that the same storage location can be used for all values r_{iu}; that is, we can replace all variables r_{iu} with a variable r. ∎

Corollary 6.22.1 *Given a two-level graph, represented by vertices V and realization f, Algorithm kTRANS generates all arcs in the transitive reduction (V, R) of the graph's oriented complement in $O(V \log V + R)$ time.*

6.3.5 Weighted Cliques and Independent Dominating Sets

We are now ready to compute a minimum weight independent dominating set in a k-level graph. An independent set is dominating if and only if it is maximal (Observation 4.30 in Chapter 4), and a set is maximal independent if and only if it is a maximal clique in the complement. We will therefore develop an algorithm that finds a minimum weight maximal clique in the complement of the k-level graph. To do so, we will simply implement Algorithm MWMC-C, minimum weight maximal clique for comparability graphs from Table 4.34, to run in $O(V^2)$ time, given the structure developed in this section. This solves the domination problem since the graph representation V and f also represents the complement of the graph. For convenience, the algorithm is reproduced below.

Theorem 6.23 *Algorithm MWMC-k finds a minimum weight independent dominating set in (a minimum weight maximal clique in the complement of) a weighted k-level graph in $O(k^2 V + V^2)$ time.*

Table 6.5: **Algorithm:** MWMC-k(V, f) [Minimum Weight Maximal Clique in the complement of a k-level graph]

Input: The weighted vertices V and a realization f of a (vertex) weighted k-level graph $G = (V, E)$.
Output: A minimum weight maximal clique in $\overline{G}$.

1 $T \leftarrow$ a transitive orientation of $\overline{G}$.
2 augment T with $s = 0$ and $t = |V| + 1$.
3 $(V, R) \leftarrow$ the transitive reduction of T.
4 $C \leftarrow$ a least weight path from s to t in (V, R).
5 **return** $C \setminus \{s, t\}$.

Proof: Since every k-level graph is a cocomparability graph, Theorem 4.34 in Chapter 4 establishes the correctness of the algorithm.

Let $G = (V, E)$ be the input k-level graph, and $T = (V, \vec{E})$ be its oriented complement. Then $\vec{E}$ is a transitive orientation of $\overline{E}$ by Property 6.12. Let s be any vertex s_i, and let t be any vertex t_i in Algorithm INITIALIZE-ORIENTED-COMPLEMENT, Step 1. Then implement Steps 1, 2, and 3 with a single call to Algorithm kTRANS. This call takes $O(V \log V + k^2 V + R)$ time according to Theorem 6.22. As before (Algorithm MWMC-C in Chapter 4), Step 4 executes in $O(V + R)$ time since R is a directed acyclic graph ([CLR90], page 536–538). The theorem follows since $|R| \leq |\vec{E}| = O(V^2)$. ∎

Let us now examine some of the consequences of Theorem 6.23. The parameter k does not figure in the complexity of the weighted clique algorithm if k is fixed, since it is consumed by the constants in the big-oh notation. In particular, the algorithm on two-level graphs has a better time complexity.

Corollary 6.23.1 *Algorithm MWMC-k finds a minimum weight independent dominating set in (a minimum weight maximal clique in the complement of) a weighted two-level graph in $O(V^2)$ time.*

Remark The proof of Theorem 6.23 argues that $|R| \leq |\vec{E}| = O(V^2)$, that is, that the size of the transitive reduction is at most the number of arcs in the graph, which in turn is at most $\binom{|V|}{2}$. Is this bound tight for k-level graphs? Can there really be $\Omega(V^2)$ arcs in the transitive reduction? Unfortunately, such a situation can arise, even for one-level (indifference) graphs. The oriented complement of the indifference graph shown in Figure 6.6 is the complete bipartite graph $K_{n,n}$, where $n = |V|/2$. Clearly,

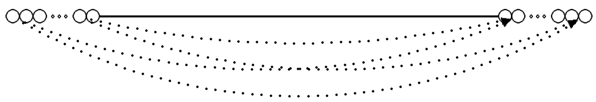


Figure 6.6: The oriented complement of this indifference graph is an oriented complete bipartite graph and is transitively reduced.

this oriented complement has n^2 arcs. Furthermore, no arc is transitively implied, so the transitive reduction also has $n^2 = |V|^2/4$ arcs.

6.4 Relation to Indifference Graphs

A one-level graph is clearly an indifference graph. A two-level graph is in some ways like two indifference graphs, one on each level. This initial observation can be misleading, though, since a two-level graph is not just the union of these two graphs. As discussed in Section 2.3.2, indifference have several characterizations and efficient algorithms for many problems. Since we are interested in these things on two-level graphs, it behooves us to study more closely how two-level graphs are related to indifference graphs.

Recall (§2.3.2) that indifference graphs are equivalently claw-free interval graphs, that is, claw-free chordal cocomparability graphs. Since two-level graphs are strip graphs, they are also cocomparability graphs. On the other hand, since C_4 is a two-level graph, two-level graphs are not chordal. Therefore they are not interval graphs. Contrast this with indifference graphs, which are in fact *unit* interval graphs.

In this section, we will see that the class of two-level graphs is:

- equivalently the intersection graphs of a set of triangles, which are themselves properly included in the class of PI^* graphs (§6.4.1),
- properly included in the class of unions of two indifference graphs (§ 6.4.3), and
- properly included in the class of intersections of two indifference graphs (§ 6.4.4).

Furthermore, we will see how to factor a two-level graph G, given its realization, into two indifference graphs whose intersection is G. We will also see that the level-edges—those that go between vertices on the same level—of a two-level graph induce an indifference graph, and that the remaining edges induce a bipartite permutation graph (§6.4.2).

6.4.1 Trapezoid Graphs and PI^* Graphs

We already know that two-level graphs are generalizations of indifference graphs, and specializations of cocomparability graphs. This section further restricts two-level graphs in a class hierarchy by demonstrating that they are specializations of trapezoid graphs. In fact, they are equivalently a restricted form of PI^*-graphs.

Trapezoid graphs were independently defined by Corneil and Kamula [CK87] and by Dagan, Golumbic, and Pinter [DGP88]. Given two horizontal lines in the plane, specify a *trapezoid* T_v by its four corners $[a_v, b_v, c_v, d_v]$ where a_v and b_v are x-coordinates on the upper line, and c_v and d_v are x-coordinates on the lower line, as shown in Figure 6.7. The intersection graph of such a trapezoid representation is called a *trapezoid graph* (or an *II graph* in [CK87]).

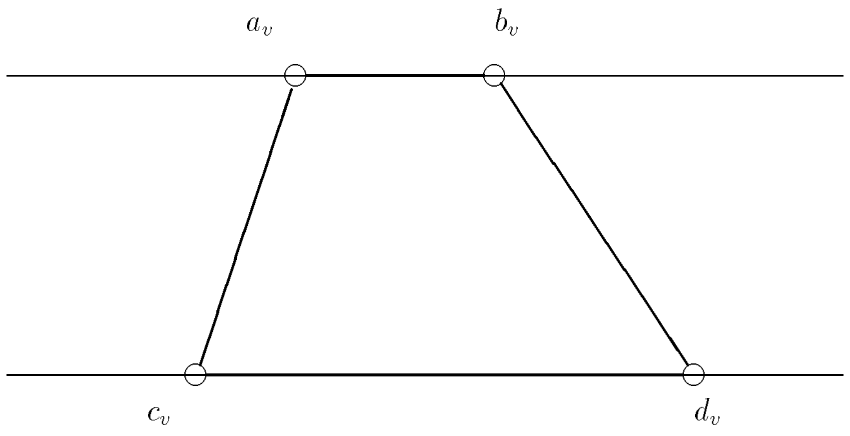


Figure 6.7: The convention used to specify a trapezoid.

Note that a trapezoid graph is an interval graph if $a_v = c_v$ and $b_v = d_v$ for all vertices v, and a permutation graph if $a_v = b_v$ and $c_v = d_v$ for all vertices v. Corneil and Kamula have also identified two other subclasses of trapezoid graphs: PI graphs, where $a_v = b_v$ for all vertices v, and PI^* graphs where $a_v = b_v$ or $c_v = d_v$ for all vertices v. The union of interval and permutation graphs is properly contained in PI graphs, which are properly contained in PI^* graphs, which are properly contained in trapezoid graphs.

In the other direction, Dagan *et al.* show that trapezoid graphs are cocomparability graphs. In fact, they prove the following theorem.

Theorem 6.24 ([DGP88]) *Trapezoid graphs are the cocomparability graphs of partially ordered sets with interval order dimension at most 2.*

The following theorem is very similar in spirit, and could really be considered a corollary of the proof presented by Dagan *et al.* Instead, the following proof is self contained and follows theirs almost verbatim.

Theorem 6.25 *A graph is a PI graph if and only if it is the cocomparability graph of the intersection of a linear order and an interval order.*

Proof: Let $L = (V, <)$ be a linear order and let $I = (V, \prec)$ be an interval order. Let $G = (V, E)$ be the cocomparability graph of $L \cap I$. Then there is a set of real numbers $\{r_v : v \in V\}$ and a set of intervals $\{I_v : v \in V\}$ such that $(u, v) \in E$ if and only if $r_u < r_v$ and $I_u \prec I_v$, or $r_v < r_u$ and $I_v \prec I_u$ (note the overloaded inequality/precedes symbols). Define the triangle (degenerate trapezoid) $T_v = [a_v, b_v, c_v, d_v]$ where $a_v = b_v = r_v$ and $[c_v, d_v] = I_v$. Then $T_u \cap T_v = \emptyset$ if and only if $u < v$ and $u \prec v$, or $v < u$ and $v \prec u$. Thus, G is a PI graph.

Conversely, let $\{T_v : v \in V\}$ be a trapezoid representation of G. For each $v \in V$, let $r_v = a_v (= b_v)$ and $I_v = [c_v, d_v]$. Since $T_u \cap T_v = \emptyset$ if and only if $r_u < r_v$ and $I_u \prec I_v$, or $r_v < r_u$ and $I_v \prec I_u$, it follows that G is the cocomparability graph of the intersection of a linear order and an interval order. ∎

Cheah's PhD thesis [Che90] shows how to recognize trapezoid graphs in $O(V^3)$ time. Cheah also mentions that the PI graph and PI^* graph recognition problems are still open, as of December 1990. Independently, Ma and Spinrad [MS91] developed an $O(V^2)$ time algorithm for recognizing trapezoid graphs. They did this by developing an $O(V^2)$ algorithm for determining if the interval dimension of a partial order is at most 2. By Theorem 6.24, a graph is a trapezoid graph if and only if it is the cocomparability graph of a partial order with interval dimension at most 2, and Ma and Spinrad's trapezoid graph recognition algorithm follows.

Triangle Graphs and Two-Level Graphs

Definition 6.26 A *triangle graph* is a trapezoid graph for which there is a trapezoid representation and a constant α such that, for all vertices v,

1. $a_v = b_v$,
2. $a_v - c_v = \alpha$, and

3. $d_v - c_v = 1$;

or

1. $c_v = d_v$,

2. $c_v - a_v = \alpha$, and

3. $b_v - a_v = 1$,

as shown in Figure 6.8. ∎

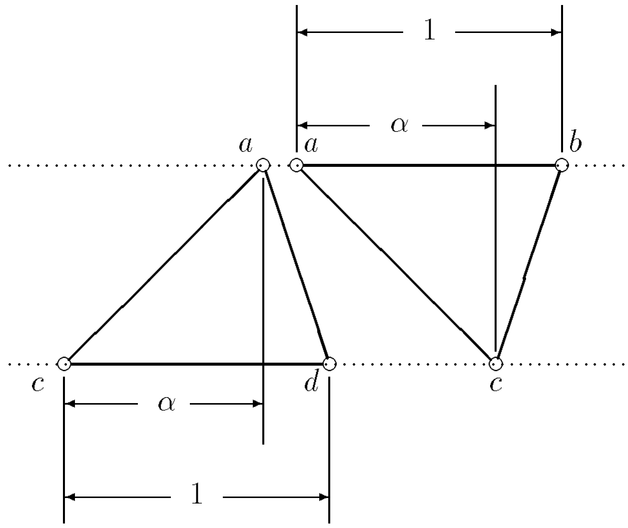


Figure 6.8: Definition of triangle graphs. All triangles in a triangle graph are congruent to an upright triangle or its reflection about the horizontal axis.

A triangle graph is called *acute* if its constant α satisfies $1/2 \leq \alpha \leq 1$.

Theorem 6.27 *A graph is a two-level graph if and only if it is an acute-triangle graph.*

Proof: Let $G = (V, E)$ be a two-level graph. Then there is a realization $f : V \to \mathbf{R} \times \{0, \tau\}$ by Definition 6.1. As before, let $\alpha = \sqrt{1 - \tau^2}$, and remember that $1/2 \leq \alpha \leq 1$.

We now represent a vertex on the lower level with the triangle on the left of Figure 6.8, and a vertex on the upper level with the triangle on the right of Figure 6.8. More formally, for each vertex v, define the representing triangle (trapezoid) $T_v = [a_v, b_v, c_v, d_v]$, where

- $c_v = x(v)$,
- $a_v = b_v = c_v + \alpha$, and
- $d_v = c_v + 1$

if $y(v) = 0$; and

- $a_v = x(v)$,
- $c_v = d_v = a_v + \alpha$, and
- $b_v = a_v + 1$

if $y(v) = \tau$. It is easy to verify that $T_u \cap T_v \neq \emptyset$ if and only if $\|f(u) - f(v)\| \leq 1$.

Conversely, let $G = (V, E)$ be an acute-triangle graph. Then there exists a constant α and a set of similar triangles $\{T_v : v \in V\}$ such that $T_u \cap T_v \neq \emptyset$ if and only $(u, v) \in E$. Set $\tau = \sqrt{1 - \alpha^2}$. Note that $0 \leq \tau \leq \sqrt{3}/2$. Assume triangle $T_v = [a_v, b_v, c_v, d_v]$ as in Figure 6.8. Now define a two-level realization $f : V \to \mathbf{R} \times \{0, \tau\}$ as follows.

$$f(v) = \begin{cases} (c_v, 0) & \text{if } a_v = b_v \\ (a_v, \tau) & \text{if } c_v = d_v \end{cases}$$

Again, it is easy to verify that $T_u \cap T_v \neq \emptyset$ if and only if $\|f(u) - f(v)\| \leq 1$. ∎

Corollary 6.27.1 *The class of two-level graphs is properly included in the class of PI^* graphs (and trapezoid graphs).*

Proof: By definition, an acute-triangle graph is also a PI^* graph and a trapezoid graph. On the other hand, $K_{1,4}$ is not a two-level graph (Lemma 6.8). However, it is a permutation graph, and therefore a PI^* graph as shown by the permutation diagram in Figure 6.9. ▮

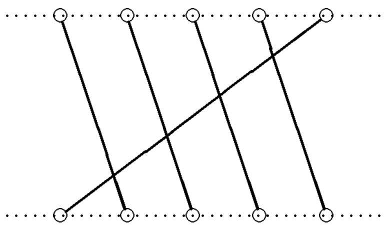


Figure 6.9: $K_{1,4}$ is a permutation graph and therefore also a PI^* graph.

Corollary 6.27.2 *Every two-level graph is the cocomparability graph of the intersection of two interval orders.*

Proof: Since every two-level graph is a trapezoid graph by Corollary 6.27.1, it is the intersection of two interval orders by Theorem 6.24. ▮

6.4.2 Cross Edges Induce a Bipartite Permutation Graph

A realization of a two-level graph will put some vertices on one level and some vertices on the other. As a result, some edges will join two vertices on the same level, and some edges will join two vertices on different levels. We want to see what kind of graph is induced by each of these groups. To this end, let $G = (V, E)$ be a two-level graph, and f be a two-level realization.

Define the *level edges* $L \subseteq E$ to be edges between vertices on the same level. That is, let

$$L = \{(u, v) : (u, v) \in E \text{ and } y_f(u) = y_f(v)\}. \tag{6.21}$$

The associated *level-edge graph* $G_L = (V, L)$ is the graph induced by these edges.

Property 6.28 *A (not necessarily connected) graph is a level-edge graph if and only if it is an indifference graph.*

Proof: An indifference graph (Definition 2.2) is trivially a level-edge graph, since it can always be realized on one level (of the two available). Conversely, let $G = (V, E)$ be a two-level graph, and f be its corresponding two-level realization. The function x_f is almost an indifference mapping for G_L, we just need to be careful to make the points corresponding to one level far enough away from those corresponding to the other. We can do this by suitably displacing one level. For example, set a displacement $\delta > 1 + M - m$, where $M = \max\{x_f(u) : y_f(u) = 0\}$ and $m = \min\{x_f(u) : y_f(u) = \tau\}$. Then the indifference realization $f_L : V \to \mathbf{R}$ suffices, where for all $v \in V$,

$$f_L(v) = \begin{cases} x_f(v) & \text{if } y_f(v) = 0 \\ x_f(v) + \delta & \text{if } y_f(v) = \tau. \end{cases}$$

∎

Define the *cross edges* X of a two-level graph to be those edges between vertices on different levels. That is, let

$$X = \{(u, v) : (u, v) \in E \text{ and } y_f(u) \neq y_f(v)\}. \tag{6.22}$$

Note that $E = X + L$. Let T identify the vertices on the top level, that is, let $T = \{v : y_f(v) = \tau\}$. Let B identify the vertices on the bottom level, that is, let $B = \{v : y_f(v) = 0\}$. The *cross-edge* graph $G_X = (T, B, X)$ induced by these edges is clearly bipartite. We will see (Corollary 6.31.1) that G_X is also a permutation graph. Again, this graph may be disconnected.

Spinrad, Brandstädt, and Stewart [SBS87] show that bipartite permutation graphs can be recognized in linear time, and that some problems that are **NP**-complete even for bipartite graphs, for example Hamilton Circuit, can be solved in polynomial time for bipartite permutation graphs. Note, however, that both bipartite and permutation graphs are perfect.

Indifference graphs and permutation graphs are incomparable: the graph shown in Figure 6.10 is an indifference graph but not a permutation graph (it is not even a compa-

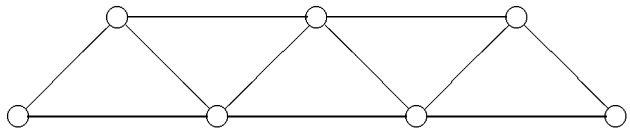


Figure 6.10: An indifference graph that is not a permutation graph

rability graph), and the claw $K_{1,4}$ is a permutation graph but not an indifference graph. Since interval graphs are chordal, and bipartite graphs have no odd cycles, a bipartite interval graph can have no cycles. Furthermore, since indifference graphs are claw free, a bipartite indifference graph can have no vertices with degree 3 or more. Therefore a bipartite indifference graph is just a set of disjoint paths, and therefore also a permutation graph. Just as clearly, not all bipartite permutation graphs are bipartite indifference graphs, the claw $K_{1,3}$ and the square C_4, for example.

There are graphs, namely the bipartite tolerance graphs[1], that are closely related to bipartite indifference graphs, and are again exactly the bipartite permutation graphs. The following definition is taken [almost] verbatim from Brandstädt, Spinrad, and Stewart [BSS87], who state that Derigs, Goetke and Schrader [DGS84] had studied this family of graphs earlier.

Definition 6.29([BSS87]) A *bipartite tolerance graph* is a bipartite graph $G = (P, Q, E)$

[1]As seems to happen often in the graph theory literature, there is a conflict of definitions here. Bipartite tolerance graphs should be considered their own class of graphs, and should not be confused with tolerance graphs [GM82] that happen to be bipartite.

for which there exists a real-valued function f on the vertices such that for all $p \in P$ [and] $q \in Q$, [the pair] $(p, q) \in E$ if and only if $|f(p) - f(q)| \leq \epsilon$ for some prespecified tolerance ϵ. ∎

Property 6.30 *A graph is a cross-edge graph if and only if it is a bipartite tolerance graph.*

Proof: Let $G_X = (V, X)$ be a cross-edge graph, and g be its corresponding two-level realization. Define a real-valued tolerance function f such that $f(v) = x_g(v)$ for every vertex $v \in V$. Then you can readily verify that $G' = (P, Q, X)$ is a bipartite tolerance graph, where

$$\begin{aligned} P &= \{v : y_g(v) = 0\}, \\ Q &= \{v : y_g(v) = \tau\}, \text{ and} \\ \epsilon &= \alpha. \end{aligned}$$

Conversely, let $G' = (P, Q, X)$ be a bipartite tolerance graph and f and ϵ be as in Definition 6.29. Pick a value for α in the interval $[1/2, 1]$ and set $x_g(v) = \alpha f(v)/\epsilon$ and $\tau = \sqrt{1 - \alpha^2}$. Then $g : V \to \mathbf{R} \times \{0, \tau\}$ is a two-level realization for $G_X = (V, X)$, where

$$g(v) = \begin{cases} (x_g(v), 0) & \text{if } v \in P \\ (x_g(v), \tau) & \text{if } v \in Q. \end{cases}$$

This is also easy to see. Two vertices on different levels of G are adjacent precisely when $|f(u) - f(v)| \leq \epsilon$, therefore precisely when $|\alpha f(u)/\epsilon - \alpha f(v)/\epsilon| \leq \alpha$, or $\|g(u) - g(v)\| \leq 1$. Hence $G_X = (V, X) = G' = (P, Q, X)$. ∎

Theorem 6.31 ([BSS87]) *Let $G = (P, Q, E)$ be a bipartite graph. Then G is a bipartite permutation graph if and only if G is a bipartite tolerance graph.*

Proof: We will prove only one direction, that G is a bipartite permutation graph if it is a bipartite tolerance graph. See [BSS87] for a proof of the converse. The following proof differs from that of Brandstädt *et al.* in that it demonstrates a connection between cocomparability graphs and bipartite tolerance graphs (and therefore cross-edge graphs). It is included in this thesis to further emphasize these connections.

Let $G = (P, Q, E)$ be a bipartite tolerance graph. A graph G is a permutation graph if and only if G and $\overline{G}$ are comparability graphs [PLE71]. All bipartite graphs are comparability graphs (orient the edges from one part to the other; this orientation is trivially transitive). Therefore G is a comparability graph.

It remains to show that $\overline{G}$ is a comparability graph. Let f be the tolerance function corresponding to G. Transitively orient $\overline{E}$ by defining the relation

$$F = \{(u, v) : (u, v) \in \overline{E} \text{ and } f(u) \leq f(v)\}.$$

To see that F is transitive, consider $(a, b) \in F$ and $(b, c) \in F$. If a, b, and c are in the same part (P or Q), then $(a, c) \in \overline{E}$, and $f(a) \leq f(b) \leq f(c)$, so $(a, c) \in F$. Otherwise, b and a are in different parts, or b and c are in different parts. If b and a are in different parts, then $f(b) - f(a) > \epsilon$, so $f(c) - f(a) = f(c) - f(b) + f(b) - f(a) > 0 + \epsilon$, and again $(a, c) \in F$. If b and c are on different levels, then a symmetric argument shows that $(a, c) \in F$. ∎

Corollary 6.31.1 *A graph is a cross-edge graph if and only if it is a bipartite permutation graph.*

Proof: Follows from Property 6.30 and the theorem. ∎

Although this corollary tells us that every cross-edge graph corresponds to two permutations, its nonconstructive nature does not tell us what these permutations are. To construct a permutation model corresponding to a cross-edge graph, let us prove the following property constructively, even thought it is an immediate consequence of Corollary 6.31.1.

Property 6.32 *The edges that go between the levels of a realization of a two-level graph induce a bipartite permutation graph.*

Proof: We will show that G_X is a bipartite permutation graph by constructing two permutations P and Q, which we will think of as lists of vertices. Construct P by starting with T in order of increasing x-coordinate. Then add vertices from B in order of *decreasing* x-coordinate. Insert each vertex from B into P immediately after its rightmost neighbour in T; see Figure 6.11, for example. Note that this process preserves the left to right order of B in P.

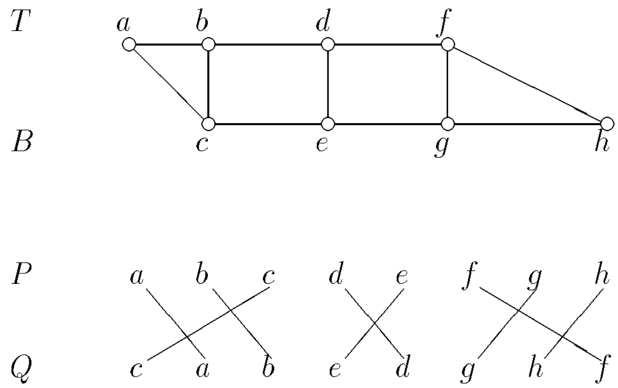


Figure 6.11: Constructing permutations P and Q from a two-level graph

Similarly, construct Q by again starting with T in order of increasing x-coordinate. But now, add vertices from B in order of *increasing* x-coordinate. Insert each vertex $v \in B$ into Q immediately before its leftmost neighbour in T. Note that this process also preserves the left to right order of B in Q.

To see that P and Q provide a permutation model for G, note first that no two vertices in T (resp. in B) are adjacent, since vertices from T (resp. from B) have the same left-to-right order in both P and Q. Now consider an edge (v_P, v_Q) in the permutation diagram corresponding to a vertex $v \in B$. By construction, this edge crosses precisely those edges corresponding to vertices between the leftmost neighbour of v and the rightmost neighbour of v inclusive. It is easy to verify (by examining the two-level realization) that these are exactly the neighbours of v. ∎

Recall from Section 1.3.1 that the union $G_1 \cup G_2$ of two graphs (binary relations) $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ on the same vertex set is the graph $G = (V, E_1 \cup E_2)$. The union is called *(edge) disjoint* if $E_1 \cap E_2 = \emptyset$.

Theorem 6.33 *Every two-level graph is the edge disjoint union of an indifference graph and a bipartite permutation graph.*

Proof: Property 6.28 and Property 6.32. ∎

6.4.3 Short Edges Induce an Indifference Graph

We will now see that every two-level graph is the union of two indifference graphs. An easy corollary of this result is that the unit-interval number (forthcoming Definition 6.36) of every two-level graph is at most two.

Let $G = (V, E)$ be a two-level τ-strip graph, and f be a corresponding two-level realization. Let S be the set of "short" edges, that is,

$$S = \{(u, v) : |x_f(u) - x_f(v)| \leq \alpha\} \tag{6.23}$$

where $\alpha = \sqrt{1 - \tau^2}$ as always. The associated *short-edge graph* $G_S = (V, S)$ is the graph induced by these edges.

Property 6.34 *A graph is a short-edge graph if and only if it is an indifference graph.*

Proof: Let $G_S = (V, S)$ be a short-edge graph, and let $G = (V, E)$ and f be the defining two-level graph and its realization. Obtain the indifference mapping f_S for G_S simply by scaling x_f. That is, $f_S(v) = x_f(v)/\alpha$. Then

$$|f_S(u) - f_S(v)| = \frac{|x_f(u) - x_f(v)|}{\alpha},$$

so that $|f_S(u) - f_S(v)| \leq 1$ if and only if $|x_f(u) - x_f(v)| \leq \alpha$, as required.

Conversely and similarly, let $G = (V, E)$ be an indifference graph and f its realization. Construct a two-level representation g for some α by scaling f and putting all the vertices on one level. That is, $g(v) = (\alpha f(v), 0)$. Then $|f(u) - f(v)| \leq 1$ if and only if $|x_g(u)/\alpha - x_g(v)/\alpha| \leq 1$ if and only if $|x_g(u) - x_g(v)| \leq \alpha$, as required. ∎

Theorem 6.35 *Every two-level graph is the union of two indifference graphs.*

Proof: Let $G = (V, E)$ be a two-level graph, and f a corresponding two-level realization. Let $G_L = (V, L)$ be the subgraph induced by edges on the same level (Equation 6.21), and let $G_S = (V, S)$ be the subgraph induced by the short edges (Equation 6.23). The graphs G_L and G_S are indifference graphs by Properties 6.28 and 6.34 respectively. It remains to show that $G = G_L \cup G_S$, that is, that $E = L \cup S$.

Clearly, $L \cup S \subseteq E$. To see that $E \subseteq L \cup S$, consider $(u, v) \in E$. Either $y_f(u) = y_f(v)$ or $y_f(u) \neq y_f(v)$. If $y_f(u) = y_f(v)$, then $(u, v) \in L$. If, on the other hand, $y_f(u) \neq y_f(v)$, then $|x_f(u) - x_f(v)| \leq \alpha$, so $(u, v) \in S$. ∎

The converse of this theorem is not true. For example, $K_{1,4}$ is not a two-level graph (Lemma 6.8), but it is the union of two edge-disjoint paths that share a common intermediate vertex. Each such path, together with the remaining isolated vertex, is an indifference graph.

A related notion is that of unit-interval number.

Definition 6.36([And88]) A *unit t-representation* of a graph assigns each vertex up to t closed unit intervals, with two vertices adjacent when any intervals assigned to them intersect. The *unit-interval number* $i_u(G)$ is the minimum value of t such that G has a unit t-representation. ▮

Andreae [And88] showed that $i_u(G) \leq \lceil \frac{1}{2}(n-1) \rceil$ for all graphs on n vertices. He also showed that the extremal graphs are stars[2] (and also C_4). Since two-level graphs are $K_{1,4}$-free, for which $i_u(G) = 2$, this might lead us to believe that $i_u(G) \leq 2$ for all two-level graphs. However, Andreae showed that the unit-interval number can grow without bound even for claw-free graphs. Nevertheless, the unit-interval number of two-level graphs *is* at most two; this is a corollary of the previous theorem.

Corollary 6.36.1 *Let G be a two-level graph. Then $i_u(G) \leq 2$, and this bound can be reached.*

Proof: Since G is the union of two indifference graphs G_L and G_S, each vertex $v \in G$ corresponds to two unit intervals, $[f_L(v), f_L(v)+1]$ and $[f_S(v), f_S(v)+1]$ respectively. Clearly, we can realize these intervals simultaneously by ensuring that the intervals for G_L are disjoint from those in G_S (with a suitable displacement, for example). The resulting set of intervals is a two-interval representation for G.

The bound is reached, for example, by C_4 and $K_{1,3}$. Both are two-level graphs for which $i_u(G) = 2$; recall that neither is an indifference graph. ▮

6.4.4 Every Two-Level Graph is the Intersection of Two Indifference Graphs

The following theorem shows that a graph has a two-level realization under the Euclidean metric if and only if it has a two-level realization under the Manhattan metric. Recall

[2] In this context, a graph of order n is *extremal* if it has unit interval number $\lceil \frac{1}{2}(n-1) \rceil$. More accurately, Andreae showed that the extremal graphs are $K_{1,n-1}$ and C_4 if n is even, graphs with induced $K_{1,n-2}$ if $n > 5$ is odd, and graphs with induced C_4 if $n = 5$.

that this property does not hold for unit disk graphs. For example, $K_{1,5}$ is an L_2 unit disk graph, but it is not an L_1 unit disk graph by the Star Lemma (Lemma 3.1). It also does not hold for τ-strip graphs. For example, $K_{1,4}$ is an L_2 strip graph (Example 5.1), but it is not an L_1 strip graph (Lemma 5.23).

Theorem 6.37 *The class of L_2 two-level graphs is equivalent to the class of L_1 two-level graphs. In particular, every L_2 two-level τ-strip graph is isomorphic to an L_1 two-level $(1-\alpha)$-strip graph.*

Proof: Let $G = (V, E)$ be an L_2 two-level graph, and let $f_2 : V \to \mathbf{R} \times \{0, \tau\}$ be a corresponding two-level realization, for some $\tau \in [0, \sqrt{3}/2]$. To avoid multiple subscripts, write $x_{f_i} = x_i$ and $y_{f_i} = y_i$ for $i = 1, 2$. Define $f_1 : V \to \mathbf{R} \times [0, 1-\alpha]$ by setting

$$f_1(v) = (x_2(v), \left(\frac{1-\alpha}{\tau}\right) y_2(v))$$

for every vertex $v \in V$. Note that $0 \leq 1 - \alpha \leq 1/2$.

To see that f_1 is an L_1 two-level realization for G, first consider two vertices u and v on the same level, so that $y_2(v) - y_2(v) = 0$. Then

$$\begin{aligned}
\|f_1(u) - f_1(v)\|_1 \leq 1 \quad &\leftrightarrow \quad |x_2(u) - x_2(v)| + \left(\frac{1-\alpha}{\tau}\right) |y_2(u) - y_2(v)| \leq 1 \\
&\leftrightarrow \quad |x_2(u) - x_2(v)| \leq 1 \\
&\leftrightarrow \quad (x_2(u) - x_2(v))^2 \leq 1 \\
&\leftrightarrow \quad (x_2(u) - x_2(v))^2 + (y_2(u) - y_2(v))^2 \leq 1 \\
&\leftrightarrow \quad \|f(u) - f(v)\|_2 \leq 1 \\
&\leftrightarrow \quad (u, v) \in E.
\end{aligned}$$

Similarly, consider two vertices u and v on different levels, so that $|y_2(u) - y_2(v)| = \tau$. Then

$$\|f_1(u) - f_1(v)\|_1 \leq 1 \quad \leftrightarrow \quad |x_2(u) - x_2(v)| + \left(\frac{1-\alpha}{\tau}\right) |y_2(u) - y_2(v)| \leq 1$$

$$
\begin{aligned}
&\leftrightarrow \quad |x_2(u) - x_2(v)| + (1-\alpha) \leq 1 \\
&\leftrightarrow \quad |x_2(u) - x_2(v)| \leq \alpha \\
&\leftrightarrow \quad \|f_2(u) - f_2(v)\|_2 \leq 1 \\
&\leftrightarrow \quad (u, v) \in E.
\end{aligned}
$$

In both cases, $\|f_1(u), f_1(v)\|_1 \leq 1$ if and only $(u, v) \in E$.

Conversely, let $G = (V, E)$ be an L_1 two-level graph, and let $f_1 : \mathbf{R} \times \{0, \epsilon\}$ be a corresponding two-level realization, where $f_1(v) = (x_1(u), y_1(v))$ for all vertices $v \in V$, and $\epsilon \in [0, 1/2]$. Then, by a similar argument, $f_2 : \mathbf{R} \times [0, 1-\alpha]$ is an L_2 two-level realization for G, where

$$
f_2(v) = (x_1(v), \left(\frac{\tau}{1-\alpha}\right) y_1(v))
$$

for every vertex $v \in V$. Here, $\alpha = 1 - \epsilon$ and $\tau = \sqrt{1-\alpha^2}$. Note that $0 \leq \tau \leq \sqrt{3}/2$, as required. ∎

Recall from Section 1.3.1 that the *intersection* $G = G_1 \cap G_2$ of two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ on the same vertex set is defined by $G = (V, E_1 \cap E_2)$. The *sphericity* (respectively *cubicity*, *boxicity*) of a graph G is the smallest k such that G is the intersection graph of a set of hyperspheres (respectively hypercubes, hyperrectangles) in k-space. Equivalently, the boxicity of a graph G is the minimum number of interval graphs whose intersection is G [Rob68b]. Similarly, the cubicity of a graph G is the minimum number of indifference graphs whose intersection is G [Rob68b]. We know, by construction, that the sphericity of two-level graphs is at most two.

A unit disk under the L_1 metric is a square rotated so that its sides are at 45 degrees to the coordinate axes. Each square is the same size; the distance from its centre to any of its corners is $1/2$ under both the L_1 and L_2 metrics. Hence we have the following corollary to Theorem 6.37.

Corollary 6.37.1 *The cubicity and boxicity of any two-level graph is at most two. These bounds can be reached.*

Proof: The bounds follow from the theorem and the preceding discussion. A two-level graph that achieves both bounds is C_4. This is not an interval graph, and therefore cannot have unit boxicity. ∎

Trapezoids

It is instructive to examine this L_1 realization, which uses intersecting L_1 disks (squares), from the following perspective. The centres of the squares lie on two levels. If two squares intersect anywhere, then they intersect inside the horizontal strip defined by these levels. With this in mind, we can create a new intersection model by clipping each square with the horizontal strip. Interestingly, the objects in the model are trapezoids, see Figure 6.12 for example. In fact, we can generate the acute-triangle realization (Theorem 6.27; see

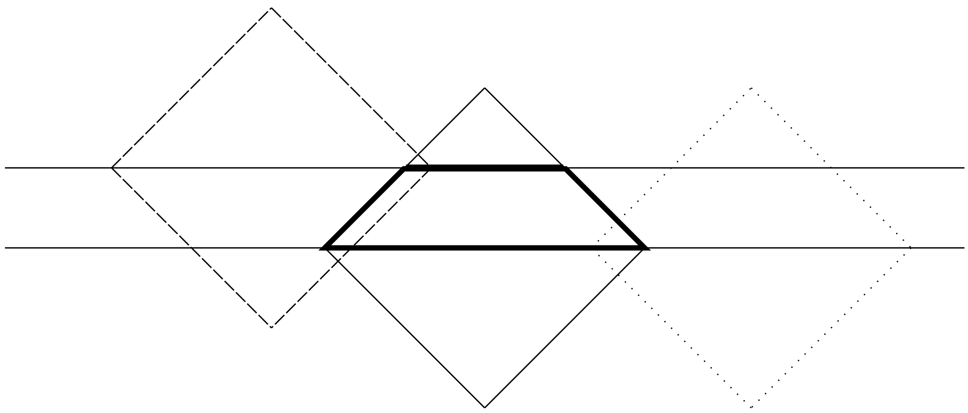


Figure 6.12: Three L_1 disks (rotated squares) corresponding to the realizations of three vertices. The disks intersect within the strip, if they intersect at all. The intersections of the squares with the strip are trapezoids.

Figure 6.8 in Section 6.4.1) as follows. First, take the thickness of the strip for the triangles to be $1 - \alpha$. Then, from the trapezoid realizing each vertex, simply remove the inside left "corner" (i.e., the second corner from the left).

Identifying the Indifference Graphs

How can we "factor" two-level graphs—construct the constituent indifference graphs—given a two-level realization? One way is to (1) construct an L_1 realization using the proof of Theorem 6.37, (2) place unit "diamonds"—rotated squares—at the realized vertices, and (3) project the sides of the diamonds onto two lines that lie at 45 degrees to the coordinate axes. Alternatively, we can use the proof of the following lemma, which, as we have seen, follows from Theorem 6.37. The proof below, however, is more direct and explicitly exhibits the constituent indifference graphs, as desired.

Property 6.38 *Every two-level graph is the intersection of two indifference graphs.*

Proof: Let $G = (V, E)$ be a two-level graph, and let $f : V \to \mathbf{R} \times \{0, \tau\}$ be a corresponding two-level realization. Specify two indifference graphs $I_0 = (V, E_0)$ and $I_\tau = (V, E_\tau)$ in terms of their indifference realizations $f_0 : V \to \mathbf{R}$ and $f_\tau : V \to \mathbf{R}$, which we now define.

Intuitively, let f_y be the x-coordinates of the two-level realization, but shift to the right (by $1 - \alpha$) all vertices with y-coordinate y. More formally, define f_0 and f_τ by the following equations.

$$f_0(v) = \begin{cases} x_f(v) + 1 - \alpha & \text{if } y_f(v) = 0 \\ x_f(v) & \text{if } y_f(v) = \tau \end{cases}$$

$$f_\tau(v) = \begin{cases} x_f(v) & \text{if } y_f(v) = 0 \\ x_f(v) + 1 - \alpha & \text{if } y_f(v) = \tau \end{cases}$$

We now need to show that $(u, v) \in E$ if and only if $(u, v) \in E_0$ and $(u, v) \in E_\tau$. We will proceed by case.

Case 1 $y_f(u) = y_f(v) = 0$.

$$(u, v) \in E \quad \leftrightarrow \quad \|f(u) - f(v)\| \leq 1$$

$$
\begin{aligned}
&\leftrightarrow \quad |x_f(u) - x_f(v)| \leq 1 \\
&\leftrightarrow \quad |x_f(u) + 1 - \alpha - (x_f(v) + 1 - \alpha)| \leq 1 \text{ and } |x_f(u) - x_f(v)| \leq 1 \\
&\leftrightarrow \quad |f_0(u) - f_0(v)| \leq 1 \text{ and } |f_\tau(u) - f_\tau(v)| \leq 1 \\
&\leftrightarrow \quad (u, v) \in E_0 \text{ and } (u, v) \in E_\tau
\end{aligned}
$$

Case 2 $y_f(u) = y_f(v) = \tau$. This case is symmetric to Case 1.

Case 3 $y_f(u) \neq y_f(v)$ and $x_f(u) \geq x_f(v)$. Without loss of generality, assume $y_f(u) = 0$ and $y_f(v) = \tau$. We first show that $(u, v) \in E$ implies $(u, v) \in E_0$ as follows.

$$
\begin{aligned}
(u, v) \in E \quad &\leftrightarrow \quad \|f(u) - f(v)\| \leq 1 \\
&\leftrightarrow \quad 0 \leq x_f(u) - x_f(v) \leq \alpha \\
&\leftrightarrow \quad 1 - \alpha \leq x_f(u) + 1 - \alpha - x_f(v) \leq 1 \\
&\leftrightarrow \quad 0 \leq 1 - \alpha \leq x_f(u) + 1 - \alpha - x_f(v) \leq 1 \\
&\leftrightarrow \quad |f_0(u) - f_0(v)| \leq 1 \\
&\leftrightarrow \quad (u, v) \in E_0
\end{aligned}
\tag{6.24}
$$

Next, we show that $(u, v) \in E$ implies $(u, v) \in E_\tau$.

$$
\begin{aligned}
(u, v) \in E \quad &\leftrightarrow \quad \|f(u) - f(v)\| \leq 1 \\
&\leftrightarrow \quad 0 \leq x_f(u) - x_f(v) \leq \alpha \\
&\leftrightarrow \quad -(1 - \alpha) \leq x_f(u) - (x_f(v) + 1 - \alpha) \leq 2\alpha - 1 \\
&\rightarrow \quad -1 \leq x_f(u) - (x_f(v) + 1 - \alpha) \leq 1 \\
&\leftrightarrow \quad |f_\tau(u) - f_\tau(v)| \leq 1 \\
&\leftrightarrow \quad (u, v) \in E_\tau
\end{aligned}
$$

Therefore, $(u, v) \in E$ only if $(u, v) \in E_0$ and $(u, v) \in E_\tau$. Conversely, if $(u, v) \in E_0$ and $(u, v) \in E_\tau$, then Equation 6.24 implies that $(u, v) \in E$.

Case 4 $y_f(u) \neq y_f(v)$ and $x_f(u) \leq x_f(v)$. This case is symmetric to Case 3.

This exhausts all cases and proves the theorem. ∎

Remark: Again, the converse of Property 6.38 does not hold. Figure 6.13 shows two indifference graphs that intersect to yield $K_{1,4}$, which is not a two-level graph by Lemma 6.8.

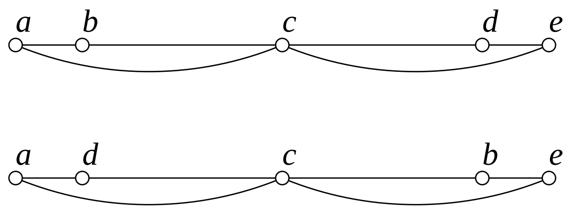


Figure 6.13: Two indifference graphs whose intersection is $K_{1,4}$.

6.5 Recognizing Two-Level Graphs

6.5.1 Striated Two-Level Graphs

It is convenient to assume that a two-level graph G has been *striated* (Definition 6.2), that is, that the stria of each vertex in G has been fixed for all realizations of G. Property 6.41 below shows that this assumption is not as restricting as it might seem. In essence, this property shows that the striae of any two-level graph can be forced by embedding it in another, highly constrained, two-level graph. This is not *strictly* true, since there are always at least two ways (corresponding to vertical reflections) to level any two level graph. These two striations, however, preserve level edges and cross edges. The following definition and observation make these notions more precise.

Definition 6.39 The *complement* $\overline{s}$ *of a two-level striation* $s : V \to \{1, 2\}$ is a function $\overline{s} : V \to \{1, 2\}$ where

$$\overline{s}(v) = \begin{cases} 1 & \text{if } s(v) = 2 \\ 2 & \text{if } s(v) = 1. \end{cases}$$

∎

Observation 6.40 *Graph* $G = (V, E, s)$ *is a striated two-level graph if and only if* $G' = (V, E, \overline{s})$ *is a striated two-level graph. Furthermore, an edge is a level-edge (respectively a cross-edge) in* $E \cup \overline{E}$ *if and only if it is a level-edge (respectively a cross-edge) in* $E' \cup \overline{E'}$.

Property 6.41 *Let* $G = (V, E, s)$ *be a striated two-level* τ*-strip graph. Then* G *is an induced subgraph of a two-level* τ*-strip graph* $G' = (V', E')$*, where every two-level* τ^**-strip realization* f' *of* G' *satisfies*

$$y_{f'}(v) = \begin{cases} 0 & \text{if } s(v) = 1 \\ \tau & \text{if } s(v) = 2 \end{cases}$$

for every vertex $v \in V$ *(or*

$$y_{f'}(v) = \begin{cases} 0 & \text{if } \overline{s}(v) = 1 \\ \tau & \text{if } \overline{s}(v) = 2 \end{cases}$$

for every vertex $v \in V$*, and for every value* $\tau^* \in [0, \sqrt{3}/2]$.

Proof: Let $G = (V, E, s)$ be a striated two-level τ-strip graph, and let f be a corresponding realization. We want to embed this graph into a larger two-level graph by following a preferred realization. The heart of the following construction is the small graph in Lemma 6.9. By the lemma, striating any of its vertices forces the striae of the others. Figuratively speaking, we want to build a horizontal "ladder" by gluing together squares ("rungs"), anchoring the ladder somewhere to the left of $f(V)$. We can then striate every vertex by fully extending the ladder to the right by n units, and doubling back to the left only partially extended, to hit the vertex in question. In particular, we can anchor the ladder at $x = 0$ and always reach $\lfloor x(v) \rfloor$ fully extended.

It remains to build ladders that reach δ in the range $0 < \delta < 1$. One way of doing so is to fully extend and partially retract the ladder by the same number n of rungs. Then the "gap" δ between the start of the ladder and its end is given by

$$\delta = n - n\alpha', \tag{6.25}$$

where α' is the uniform horizontal length of a retracted square, see Figure 6.14.

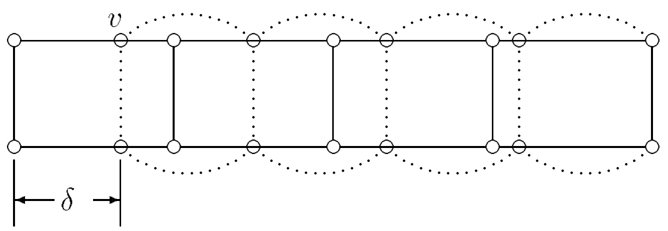


Figure 6.14: Forcing striae. The ladder hits vertex v at $x(v) = \lfloor x(v) \rfloor + \delta$. The ladder shown is extended 4 units, and retracted 4 units.

To build a ladder with a gap δ, first choose the number n of rungs. Clearly, larger values of n give us more flexibility; we just need to be sure that n is sufficiently large. Note that $\alpha' > \alpha$; otherwise the square diagonals would be too close together and hence adjacent. Therefore by Equation 6.25, it must be that

$$n = \frac{\delta}{1 - \alpha'} > \frac{\delta}{1 - \alpha}.$$

So, since $\delta < 1$, choose $n = \lceil \frac{1}{1-\alpha} \rceil$. For example, if $\alpha = 1/2$, then $n = 2$, the minimum required for stria forcing with Figure 6.3. It is now easy to set α' to $1 - \frac{\delta}{n}$ (by Equation 6.25), which is between 1/2 and 1, as required, since $n \geq 2$ and $0 < \delta < 1$.

Construct the graph G' by building $|V|$ ladders, all anchored at the same point, to fix the stria of every vertex in G. The vertices of G' are those of V plus those of the ladders.

The edges of G' are determined by the realization just constructed. Note that this construction may create many more edges than those of G and the ladders. Nevertheless, G is an induced subgraph of G', and so is the "double rung" configuration (Figure 6.3) for every ladder. Any realization of G' will therefore have one of two striations; its restriction to G will have striation s or $\overline{s}$. ▮

6.5.2 Orienting the Complement of a Two-Level Graph

Every transitive orientation of the complement of a two-level graph is compatible[3] with some two-level realization. To prove this, given a realization and an implication class of the graph's complement, we will construct a new realization after reversing the implication class. To do so, we "rotate" the implication class. This rotation:

1. preserves all adjacencies (Lemma 6.59),
2. preserves all arcs not in the implication class (Lemma 6.58), and
3. reverses all arcs in the implication class (Lemma 6.57).

Therefore, given a compatible orientation, another can be obtained by reversing any implication class. This proof of the following lemma expands on this argument. The notation for comparing x-coordinates can obscure the vertices. Therefore, abbreviate inequalities of the form $x_f(a) + \alpha \leq x_f(b)$ as $a + \alpha \overset{f}{\leq} b$. The proof of Lemma 6.42 uses Lemmas 6.57, 6.58, and 6.59; these are proved later in this subsection.

Lemma 6.42 *Let $G = (V, E)$ be a two-level τ-strip graph with a chordless dominating path of three or more edges. Then every union $\vec{E} = \bigcup_i A_i$ of implication classes satisfying $\vec{E} \cap \vec{E}^{-1} = \emptyset$ and $\vec{E} + \vec{E}^{-1} = \overline{E}$ is compatible with some two-level τ-strip realization. Furthermore, every such union $\vec{E}$ is a transitive orientation of $\overline{E}$.*

[3] Recall that an orientation $\vec{E}$ of (the complement of) a two-level graph $G = (V, E)$ is *compatible* with a two-level realization $f : V \to \mathbf{R} \times \{0, \tau\}$ if $(u, v) \in \vec{E}$ if and only if $(u, v) \in \overline{E}$ and $x_f(u) < x_f(v)$.

Proof: Let $f : V \to \mathbf{R} \times \{0, \tau\}$ be a two-level realization of $G = (V, E)$. Now consider any union $\vec{E} = \bigcup_i A_i$ of implication classes satisfying $\vec{E} \cap \vec{E}^{-1} = \emptyset$ and $\vec{E} + \vec{E}^{-1} = \overline{E}$. Let $(s, \ldots, t)$ be a chordless dominating path of three or more edges where $s \overset{f}{<} t$. If $(s,t) \notin \vec{E}$, then $(t,s) \in \vec{E}$, so reflect f about the vertical axis. Then rotate every implication class $[(u,v)]$ for which $(u,v) \in \vec{E}$ but $v \overset{f}{<} u$. The graph $G' = (V, E')$ generated by the resulting points is a two-level graph by definition. By repeated applications of Lemma 6.57, Lemma 6.58, and Lemma 6.59, $E' = E$ and $\vec{E}$ is the orientation of $\overline{E}$ that is compatible with the rotated realization. Furthermore, $\vec{E}$ is a transitive orientation by Lemma 3.6. ▮

Theorem 6.43 *Every transitive orientation of a two-level graph is compatible with some two-level realization.*

Proof: Every transitive orientation satisfies the conditions of the previous lemma by Theorem 4.51. ▮

Note that Lemma 6.42 does *not* hold for arbitrary cocomparability graphs. That is, some unions of the implication classes of some cocomparability graphs are *not* transitive. For example, the complement of a triangle is a cocomparability graph, and every edge in the complement is its own colour class (Definition 4.54). Nevertheless, the non-edges cannot be arbitrarily oriented, since two of the eight possible orientations are not transitive ([Gol80], page 107). We can easily construct a cocomparability graph G that contains such a triangle in the complement, even though G has an arbitrarily long chordless dominating path. Begin with a long path $(s, \ldots, t)$, and make three new vertices $\{a, b, c\}$ adjacent only to a vertex near the middle of the path. It is easy to verify that the vertices $\{a, b, c\}$ induce a triangle in $\overline{G}$, and that none of these complementary edges is forced by (s, t).

The Neighbourhood of an Unextendable Captured Arc

Let $G = (V, E)$ be a two-level τ-strip graph for some $\tau \in [0, \sqrt{3}/2]$, and let $f : V \to \mathbf{R} \times \{0, \tau\}$ be a two-level realization. As usual for two-level graphs, let $\alpha = \sqrt{1 - \tau^2}$. Let $\vec{G} = (V, \vec{E})$ be the orientation of the complement of G that is compatible with f. Let us assume in this report that G is large enough to have a chordless dominating path $(s, \ldots, t)$ with at least three edges, where $s \overset{f}{<} t$. Then by the Capture Theorem (Theorem 4.57), every arc in $\vec{E}$ is either *forced* or *captured* by $(s, t) \in \vec{E}$. Clearly, every forced arc is in the implication class $[(s, t)]$. Just as clearly, the implication class of every captured arc contains only captured arcs.

Lemma 6.44 *Every arc $(u, v) \in \vec{E}$ captured by (s, t) is a cross-edge. Furthermore, vertices u and v have at least one common neighbour.*

Proof: By Definition 4.56, (u, v) is part of a square or a claw in which all other arcs are forced by (s, t). In both cases, u and v have a common neighbour in G. Furthermore, if (u, v) is part of a square, then it is a cross-edge by Lemma 6.6. On the other hand, if (u, v) is part of a claw oriented as in Figure 4.15 (see Definition 4.56), then it is a cross-edge by Lemma 6.7. That is, (u, v) is a cross-edge in either case. ∎

Definition 6.45 Say that an arc $(u, v) \in \vec{E}$ is *extendable to the left* if u has a left neighbour on its level, that is, if there exists an edge $(u', u) \in E$ where $u' \overset{f}{<} u$ and $y_f(u') = y_f(u)$ (note that $(u', v) \in \vec{E}$). Similarly, an arc $(u, v) \in \vec{E}$ is *extendable to the right* if v has a right neighbour on its level, that is, if there exists an edge $(v, v') \in E$ where $v \overset{f}{<} v'$ and $y_f(v) = y_f(v')$ (note that $(u, v') \in \vec{E}$). An arc that is extendable neither to the left nor to the right is called *unextendable*. ∎

Lemma 6.46 *There is an unextendable arc in the implication class of every arc.*

Proof: Let $[i]$ be the implication class of some arc i. Now let (a,b) be the arc with least value $x_f(a)$ in $[i]$ (break ties by selecting the arc with greatest value $x_f(b)$). Then (a,b) is not extendable to the left for, if it were, there would exist an arc (a',b) forced by (a,b) with $a' \overset{f}{<} a$. Similarly, (a,b) is not extendable to the right for, if it were, there would exist an arc (a,b'), forced by (a,b), with $b \overset{f}{<} b'$. ∎

In particular by the previous lemma, there is an unextendable arc in the implication class of every *captured* arc. Let (a,b) be an unextendable captured arc with $y_f(a) = 0$ and $y_f(b) = \tau$, as shown in Figure 6.15. A symmetric argument holds if $y_f(a) = \tau$ and $y_f(b) = 0$.

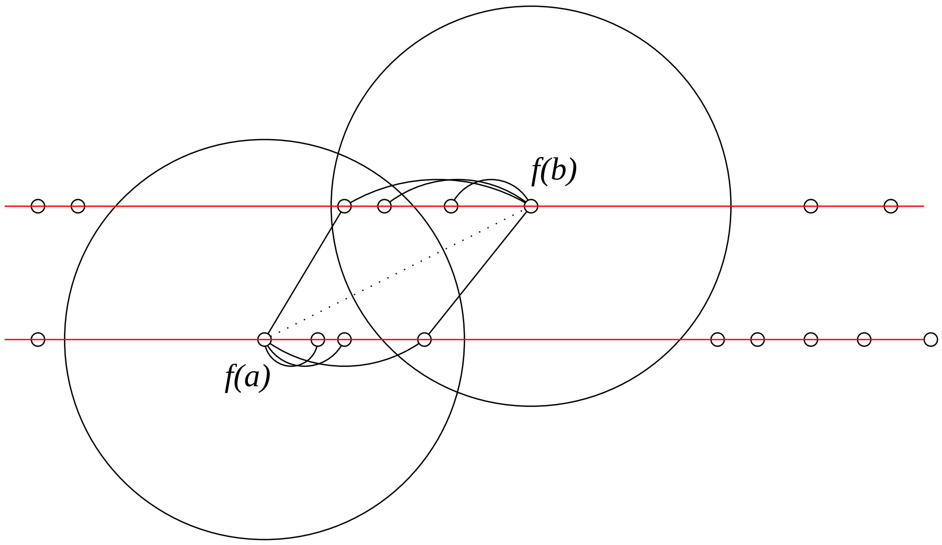


Figure 6.15: The arc (a,b) is unextendable. For clarity, only edges from a and b are shown.

Observation 6.47 *The unit-radius disk about $f(a)$ intersects the unit-radius disk about $f(b)$ on both the upper and lower levels.*

Proof: Vertices a and b share at least one neighbour by Lemma 6.44. The realization of this neighbour lies on one level or the other, so the disks intersect on that level. If the disks intersect on one level, then they intersect on the other. ∎

Let the unit-radius circle about a intersect the upper level at a_{nw} and a_{ne}, where $a_{nw} \overset{f}{<} a_{ne}$, and the lower level at a_{sw} and a_{se}, where $a_{sw} \overset{f}{<} a_{se}$, as shown in Figure 6.16. Similarly, let the unit-radius circle about b intersect the levels at $b_{nw} \overset{f}{<} b_{ne}$ and $b_{sw} \overset{f}{<} b_{se}$.

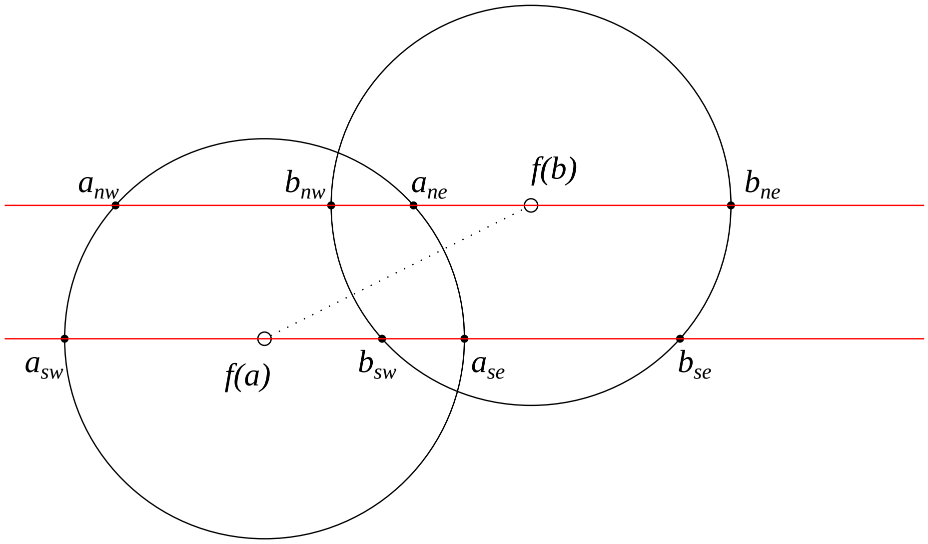


Figure 6.16: The unit-radius disks about a and b intersect the levels at four distinct points each.

More analytically,

$$\begin{aligned}
a_{nw} &= f(a) + (-\alpha, \tau), \\
a_{ne} &= a + (\alpha, \tau), \\
a_{sw} &= f(a) + (-1, 0), \\
a_{se} &= f(a) + (1, 0), \\
b_{nw} &= f(b) + (-1, \tau), \\
b_{ne} &= f(b) + (1, \tau), \\
b_{sw} &= f(b) + (-\alpha, -\tau), \\
b_{se} &= f(b) + (\alpha, -\tau).
\end{aligned}$$

The following observations are straightforward.

Observation 6.48 *Point $f(a)$ is within unit distance of point $p = (x_p, \tau)$ if and only if $p \in [a_{nw}, a_{ne}]$, and $f(a)$ is within unit distance of point $q = (x_q, 0)$ if and only if and $q \in [a_{sw}, a_{se}]$. Therefore vertex a is adjacent to vertex v if and only if v satisfies $f(v) \in [a_{nw}, a_{ne}]$ or $f(v) \in [a_{sw}, a_{se}]$. Similarly, Point $f(b)$ is within unit distance of point $p = (x_p, \tau)$ if and only if $p \in [b_{nw}, b_{ne}]$, and $f(b)$ is within unit distance of point $q = (x_q, 0)$ if and only if and $q \in [b_{sw}, b_{se}]$. Therefore vertex b is adjacent to vertex v if and only if v satisfies $f(v) \in [b_{nw}, b_{ne}]$ or $f(v) \in [b_{sw}, b_{se}]$.*

Observation 6.49 $\|a_{ne} - a_{nw}\| \geq 1$ *and* $\|b_{se} - b_{sw}\| \geq 1$.

Proof: The radii of the circles about $f(a)$ and $f(b)$ is 1, and the distance τ between levels is at most $\sqrt{3}/2$. ∎

Observation 6.50 $a_{nw} \overset{f}{<} b_{nw} \overset{f}{\leq} a_{ne} \overset{f}{<} f(b) \overset{f}{<} b_{ne}$ *and* $a_{sw} \overset{f}{<} f(a) \overset{f}{<} b_{sw} \overset{f}{\leq} a_{se} \overset{f}{<} b_{se}$.

Proof: The unit-radius disks about a and b intersect each other on the upper level, so $b_{nw} \overset{f}{\leq} a_{ne}$. But a and b are not adjacent, so $a_{ne} \overset{f}{<} b$. Therefore Observation 6.49 implies $a_{nw} \overset{f}{<} b_{nw}$. The proof of the second set of inequalities is similar. ∎

Observation 6.51 *The intervals $[a_{sw}, f(a))$ and $(f(b), b_{ne}]$ are "empty". That is, no vertex $v \in V$ satisfies $f(v) \in [a_{sw}, f(a))$ or $f(v) \in (f(b), b_{ne}]$.*

Proof: A vertex $v \in V$ that satisfies $f(v) \in [a_{sw}, f(a))$ or $f(v) \in (f(b), b_{ne}]$ would contradict the nonextendability of (a, b). ∎

Observation 6.52 *The intervals $[a_{nw}, b_{nw})$ and $(a_{se}, b_{se}]$ are "empty". That is, no vertex $v \in V$ satisfies $f(v) \in [a_{nw}, b_{nw})$ or $f(v) \in (a_{se}, b_{se}]$.*

Proof: If $v \in V$ satisfies $f(v) \in [a_{nw}, b_{nw})$, then $(a, v) \in E$ and $(v, b) \in \overline{E}$ by Observation 6.48. Therefore level-arc (v, b) would directly force (a, b), contradicting Lemma 6.44. The proof for interval $(a_{se}, b_{se}]$ is similar. ∎

Definition 6.53 The *lower (proper) neighbourhood* L_a of a is the interval (of points in the plane) $L_a = [f(a), b_{sw})$. Similarly, the *upper (proper) neighbourhood* U_b of b is the interval $U_b = (a_{ne}, b]$. See Figure 6.17 for an illustration. ∎

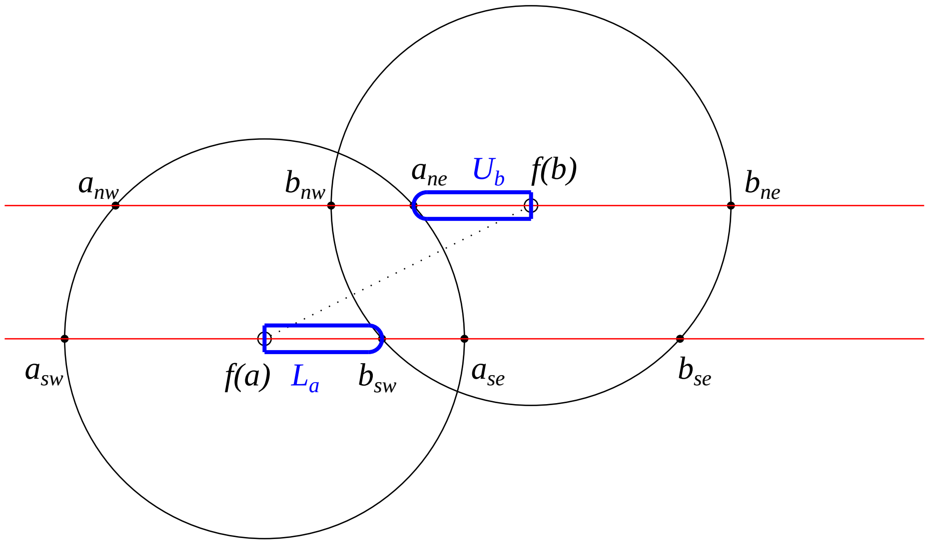


Figure 6.17: The unit-radius disks about a and b defines the neighbourhoods L_a and U_b.

Lemma 6.54 *The implication class of (a, b) is equivalent to the set of arcs that go from L_a to U_b. More formally, $[(a, b)] = \{(u, v) : (u, v) \in \vec{E},\ f(u) \in L_a,$ and $f(v) \in U_b\}$.*

Proof: Suppose $(a, b) = (a_0, b_0)$. If $(u, v) \in \vec{E}$, $f(u) \in L_a = [f(a), b_{sw})$, and $f(v) \in U_b = (a_{ne}, f(b)]$, then $(a, b)\Gamma(a, v)\Gamma(u, v)$ is a forcing chain by Observation 6.48. It follows that $(u, v) \in [(a, b)]$.

On the other hand, we will prove by induction on k, that if

$$(a_0, b_0)\Gamma(a_1, b_1)\Gamma \cdots \Gamma(a_k, b_k)$$

is a forcing chain, then $(a_k, b_k) \in \vec{E}$, $f(u) \in L_a$, and $f(v) \in U_b$. The claim is trivially true for $k = 0$, so assume the claim is true for some $k \geq 0$ and consider $k+1$. By Lemma 6.44, all arcs in $[(a_0, b_0)]$ are cross arcs. Therefore $y_f(a_i) = y_f(a_0) = 0$ and $y_f(b_i) = y_f(b_0) = \tau$ for all $i \in [0, k]$. By the definition of the forcing relation (Definition 4.49), $(b_0, \ldots, b_k)$ is a path in G. It follows that

$$b_k \overset{f}{\leq} b \tag{6.26}$$

(otherwise b would have a neighbour to its right on the upper level). Similarly $a \overset{f}{\leq} a_k$. Furthermore $a_k + \alpha \overset{f}{<} b_k$, since $(a_k, b_k) \in \vec{E}$, and therefore

$$a_{ne} \overset{f}{=} a + \alpha \overset{f}{\leq} a_k + \alpha \overset{f}{<} b_k. \tag{6.27}$$

By Equations 6.26 and 6.27, $a_{ne} \overset{f}{<} b_k \overset{f}{\leq} b$, so that $f(b_k) \in U_b$. By a similar argument $f(a_k) \in L_a$. ∎

The Rotation Operation

We are now ready to describe our rotation operation. Create a new realization r from the realization f and the captured unextendable arc (a, b) by rotating $L_a \cup U_b$ by 180 degrees about the midpoint $m = (a + b)/2$. Note that this is equivalent to "reflecting" each point through the *point* m. More formally

$$r(v) = \begin{cases} f(a) + f(b) - f(v) & \text{if } v \in L_a \cup U_b, \\ f(v) & \text{otherwise} \end{cases}$$

for all $v \in V$. The realization r satisfies the required properties, as follows.

Definition 6.55 Define $G' = (V, E')$ to be the graph generated by the points $r(V)$, and $\vec{G'} = (V, \vec{E'})$ to be the orientation compatible with r. That is, let $E' = E(G_\tau(r(V)))$, and let $\vec{E'} = \{(u, v) : (u, v) \in E' \text{ and } u \overset{f}{<} v\}$. ∎

Observation 6.56 *The rotation operation maps the intervals L_a and U_b into each other. More to the point, if $v \in L_a$, then $x(a_{ne}) < x_r(v) \leq x_f(b)$, and if $v \in U_b$, then $x_f(a) \leq x_r(v) < x(b_{sw})$.*

Lemma 6.57 *The rotation operation reverses all arcs in the implication class $[(a,b)]$. More formally $[(a,b)]^{-1} \subseteq \vec{E}'$.*

Proof: If $(u,v) \in [(a,b)]$ then $f(u) \in L_a$ and $f(v) \in U_b$ by Lemma 6.54. Since the rotation operation rotates both points 180 degrees about m, it reverses the vector from $f(u)$ to $f(v)$. More formally,

$$\begin{aligned} g(v) - g(u) &= f(a) + f(b) - f(v) - [f(a) + f(b) - f(u)] \\ &= -[f(v) - f(u)]. \end{aligned}$$

∎

Lemma 6.58 *The rotation operation preserves all arcs that are not in the implication class $[(a,b)]$. More formally, $\vec{E} \setminus [(a,b)] \subseteq \vec{E}'$.*

Proof: Consider any arc $(u,v) \in \vec{E} \setminus [(a,b)]$. Then $f(u)$ and $f(v)$ are not both in $L_a \cup U_b$, by Lemma 6.54. If neither $f(u)$ nor $f(v)$ are in $L_a \cup U_b$, then $r(u) = f(u)$ and $r(v) = f(v)$, so again $(u,v) \in \vec{E}'$. Therefore assume without loss of generality that $f(u) \in L_a$ but $f(v) \notin L_a \cup U_b$ (the arguments for $f(u) \in U_b$, and for $f(u) \notin L_a \cup U_b$ but $f(v) \in L_a \cup U_b$, are symmetric). Note that $r(u) = f(a) + f(b) - f(u)$ and that $r(v) = f(v)$. First suppose that $y_f(v) = 0$ so that (u,v) is a (forced) level arc as shown in Figure 6.18. It cannot be that v is adjacent to b for, if it were, (a,b) would be forced by the chain $(u,v)\Gamma(a,v)\Gamma(a,b)$. Therefore $b_{se} \overset{f}{<} v$ since, in addition, $f(v) \notin L_a$. Since $r(u) \in (a_{ne}, f(b)]$ and $f(b)$ is exactly unit distance from b_{se}, it follows that $r(v) = f(v)$ is more than unit distance to the right of $r(u)$.

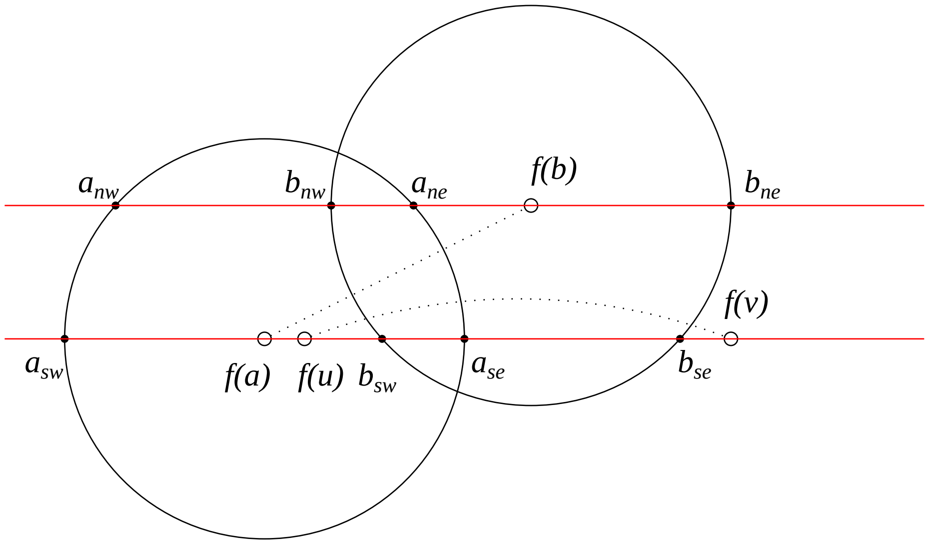


Figure 6.18: The rotation operation preserves arcs with v on the lower level.

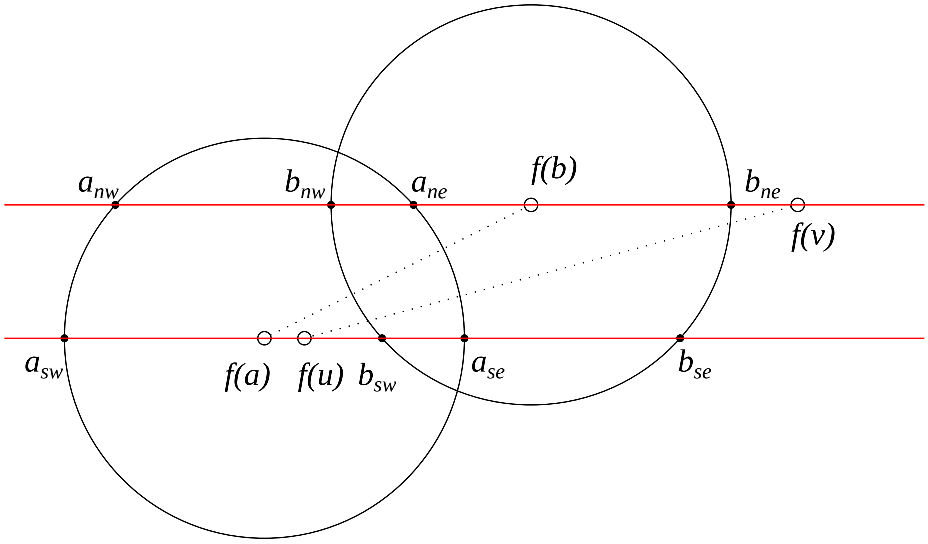


Figure 6.19: The rotation operation preserves arcs with v on the upper level.

Now suppose $y_f(v) = \tau$ as shown in Figure 6.19. It cannot be that $v \overset{f}{\leq} a_{ne}$ since then $u \overset{f}{<} v \overset{f}{\leq} a_{ne}$ would imply that $(u, v) \in E$, a contradiction. Since in addition $v \notin U_b$, it must be that $b \overset{f}{<} v$, so that $b+1 \overset{f}{<} v$ since b has no right neighbours on this level. Since $r(u) \in (a_{ne}, f(b)]$, it follows that $x_r(u) + 1 \leq x_f(b) + 1 < x_r(v)$ so that $(u, v) \in \vec{E}'$. ∎

Lemma 6.59 *The rotation operation preserves all edges. More formally, $E \subseteq E'$.*

Proof: Consider any edge $(u, v) \in E = E(G_\tau(f(V)))$. If $f(u)$ and $f(v)$ are both in $L_a \cup U_b$, then $(u, v) \in E'$ since the rotation operation preserves distances so that $\|r(u) - r(v)\| = \|f(u) - f(v)\| \leq 1$. Similarly, if neither $f(u)$ nor $f(v)$ are in $L_a \cup U_b$, then $r(u) = f(u)$ and $r(v) = f(v)$ so that again $(u, v) \in E'$. Therefore assume without loss of generality that $f(u) \in L_a$ but $f(v) \notin L_a \cup U_b$ (the arguments for $f(u) \in U_b$, and for $f(u) \notin L_a \cup U_b$ but $f(v) \in L_a \cup U_b$, are symmetric).

Since $(u, v) \in E$, the point $f(v)$ is within unit distance of $f(u) \in L_a = [f(a), b_{sw})$. Therefore $f(v) \in [a_{sw}, b_{se}]$ (by Observation 6.49) or $f(v) \in [a_{nw}, f(b)]$ But the intervals $[a_{sw}, f(a))$, $(a_{se}, b_{se}]$, and $[a_{nw}, b_{nw})$ are all empty by Observations 6.51 and 6.52. Furthermore $f(v) \notin L_a \cup U_b$. It follows that either $f(v) \in [b_{sw}, a_{se}]$ or $f(v) \in [b_{nw}, a_{ne}]$. In either case $f(v)$ is within unit distance of $f(b)$.

First suppose that $f(v) \in [b_{sw}, a_{se}]$ as shown in Figure 6.20. Then $f(v)$ is also within unit distance of a_{ne}. To see this, recall that $x(a_{ne}) = x_f(a) + \alpha$ and $x(a_{se}) = x_f(a) + 1$. Therefore

$$x(a_{se}) - x(a_{ne}) = 1 - \alpha \leq \alpha$$

since $1/2 \leq \alpha \leq 1$, and and so a_{se} is within unit distance of a_{ne}. Therefore both a and a_{se} are within unit distance of a_{ne}. It follows that $f(v)$ is within unit distance of a_{ne} by the convexity of the unit disk about a_{ne}. Hence both $f(b)$ and a_{ne} are within unit distance of $f(v)$. Consequently $r(u) \in (a_{ne}, b]$ is within unit distance of $r(v) = f(v)$ by the convexity of the unit-radius disk about $f(v)$. It follows that $(u, v) \in E'$.

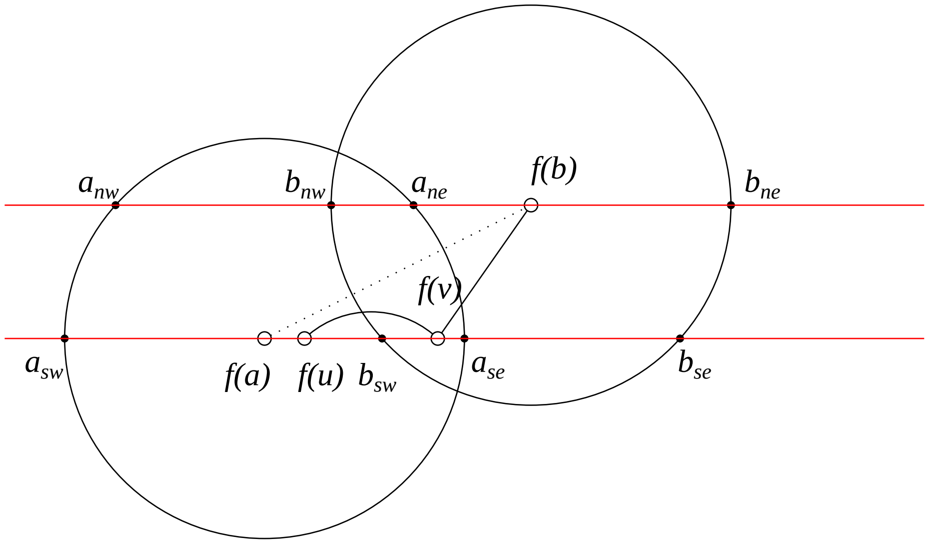


Figure 6.20: The rotation operation preserves edges with v on the lower level.

Now suppose $f(v) \in [b_{nw}, a_{ne}]$ as shown in Figure 6.21. It follows that $r(v) = f(v)$ is within unit distance of $r(u) \in U_b = (a_{ne}, b]$ since $b_{nw} \overset{r}{<} v \overset{r}{\leq} a_{ne} \overset{r}{<} u \overset{r}{\leq} f(b)$ and b_{nw} and $f(b)$ are exactly unit distance apart. It follows that $(u, v) \in E'$. ▮

Corollary 6.59.1 $E = E'$.

Proof: Since rotating twice is the identity operator, the lemma implies that $E' \subseteq E$. Note that this also follows from Lemmas 6.57 and 6.58. ▮

Orienting the Complement of a Striated Two-Level Graph

So far in this subsection, we have seen that every transitive orientation of the complement of a two-level graph is compatible with some realization. This is no longer true if we restrict the stria of the vertices. So let us consider striated two-level graphs, and let us continue to assume that they have dominating paths with at least three edges. Every realization of such a graph is compatible with one of only two complement orientations, which are in fact mutual inverses. To prove this, it is convenient to think of an arc

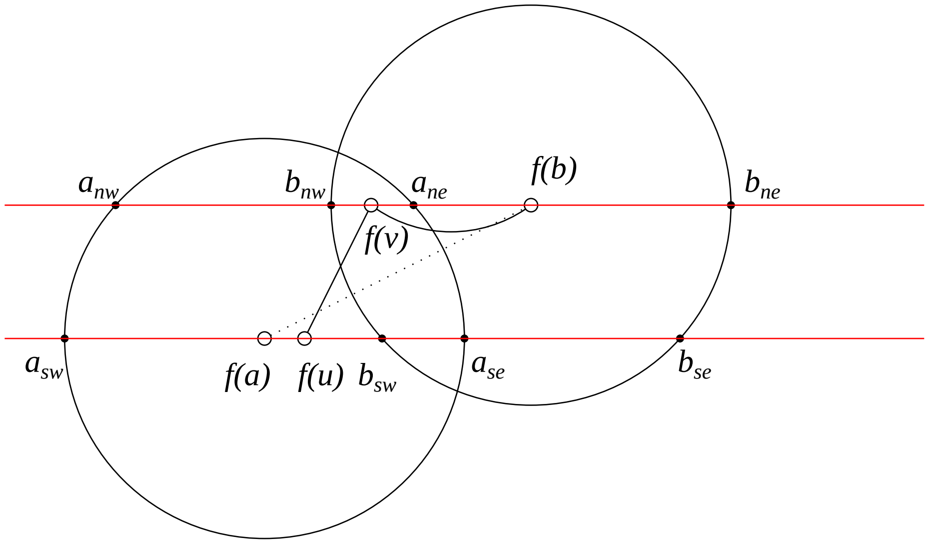


Figure 6.21: The rotation operation preserves edges with v on the upper level.

determining another if they have the same orientation in every compatible realization. The following definition makes this notion more precise.

Definition 6.60 Let $G = (V, E)$ be a two-level graph. Say that an arc $(a, b) \in \overline{E}$ *determines (the orientation of)* another arc $(c, d) \in \overline{E}$ if $x_f(a) < x_f(b)$ if and only if $x_f(c) < x_f(d)$ for every realization f of G. ∎

Theorem 6.61 *Let $G = (V, E, s)$ be a connected, striated two-level graph, and let $P = (p_0, \ldots, p_k)$ be a chordless dominating path, with at least three edges ($k \geq 3$), in G. Then (p_0, p_k) determines (a, b), or (p_0, p_k) determines (b, a), for every ordered pair $(a, b) \in \overline{E}$.*

Proof: The left-to-right order under any realization is a transitive order for $\overline{E}$; let us write $u < v$ if $x_f(u) < x_f(v)$. Therefore, if (p_0, p_k) transitively forces (a, b), it must be that (p_0, p_k) determines (a, b). Similarly, if (p_0, p_k) forces (b, a), it must be that (p_0, p_k) determines (b, a).

So assume that (p_0, p_k) forces neither (a,b) nor (b,a). By Theorem 4.57, (p_0, p_k) captures (a,b), that is, (a,b) is part of one of the three induced graphs in Figure 6.22, where the directed edges are forced by (p_0, p_k).

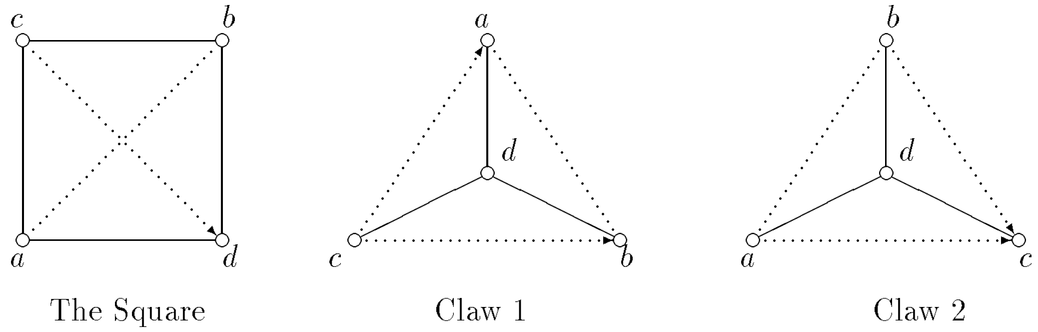


Figure 6.22: Two-level graphs that are not forced. If neither (a,b) nor (b,a) is forced by (p_0, p_k), then (a,b) is part of one of these three induced subgraphs.

Suppose first that (a,b) is part of the Square (a,c,b,d,a) in the figure. If $a < b$ for some realization, then (b,c) and (a,d) are level edges, and the rest cross edges, by Lemma 6.6. On the other hand, if $b < a$, then (b,d) and (c,a) are level edges, and the rest cross edges, by the same lemma. Only one possibility is consistent with striation s, so this possibility must hold for any realization f consistent with s. It follows that (p_0, p_k) determines (a,b).

Now suppose that (a,b) is a part of Claw 1 in the figure. If $a < b$, then (c,d) and (d,b) are level edges, the rest are cross edges, by Lemma 6.7. On the other hand, if $b < a$, then (c,d) and (d,a) are level edges, the rest cross edges, by the same lemma. Again, only one possibility is consistent with s, and (p_0, p_k) determines (a,b).

Finally, suppose that (a,b) is a part of Claw 2 in the figure. If $a < b$, then (a,d) and (d,c) are level edges, the rest are cross edges, by Lemma 6.7. On the other hand, if $b < a$, then (b,d) and (d,c) are level edges, the rest cross edges, by the same lemma. Again, only one possibility is consistent with s, and (p_0, p_k) determines (a,b). ∎

Algorithms for Orienting Two-Level Graphs

By virtue of Theorem 6.43, any transitive orientation of the complement of a two-level graph is compatible with some realization. A transitive orientation can be found in $O(V^2)$ time using Spinrad's algorithm [Spi85].

On the other hand, there are only two transitive orientations of a striated two-level graph that are compatible with some realization. These orientations, too, can be found in polynomial time, given the endpoints (p_0, p_k) of a chordless dominating path with at least three edges. The following algorithm finds the orientation containing the directed arc (p_0, p_k) in $O(V^4)$ time. First enumerate the implication class $[(p_0, p_k)]$ in $O(V^3)$ time[4] ([Gol80] pages 129–132). That is, determine which edges in $\overline{E}$ are forced by (p_0, p_k). Then for every unforced edge $(a, b) \in \overline{E}$, find a claw or square that captures it. This can be done in $O(V^2)$ time by examining all distinct pairs c, d of vertices in V, and testing in constant time whether $\{a, b, c, d\}$ induces a capturing square or claw. Since the capturing claw or square is striated, it determines the orientation of the edge (a, b), as we demonstrated in this section. Therefore all unforced edges can be oriented in $|\overline{E}| \cdot V^2 = O(V^4)$ time.

6.5.3 There is No Forbidden-Ordered-Triple Characterization of Striated Two-Level Graphs

The class of graphs characterized by a family F of forbidden linearly-ordered subgraphs is the set of graphs G whose vertices can be linearly ordered such that no graph in F is an induced linearly ordered subgraph of G. A *(linearly) ordered triple* is a linearly-ordered graph (Definition 4.4) with three vertices. Many families of graphs, including indifference

[4]Although Spinrad's algorithm [Spi85] will transitively orient a comparability graph in $O(V^2)$ time, it is not clear if it can be used to enumerate implication classes. Certainly it is not easy to enumerate *all* implications classes using Spinrad's algorithm since such an algorithm could be used to recognize comparability graphs by verifying that no arc and its reversal are in the same implication class. However, Spinrad's algorithm for recognizing comparability graphs relies on matrix multiplication, which results in a lower bound of $\Omega(M(V))$ on the execution time.

graphs, interval graphs, and cocomparability graphs, have been characterized in terms of sets of forbidden ordered triples (*cf.* [Dam90]). In this subsection we will see that two-level graphs have no forbidden ordered triple characterization. This is especially interesting since two-level graphs are generalizations of indifference graphs and specializations of cocomparability graphs.

Assume that all two-level graphs in this subsection, including triples, are striated. Let us say that an ordered triple is *forbidden* for strip thickness τ if it is not a two-level τ-strip graph. We can then say that a linear order for a graph is *unforbidden* if it does not contain any forbidden ordered triples. Now consider the unforbidden order shown in Figure 6.23. The linearly-ordered graph shown in this figure is not an ordered two-level graph (since

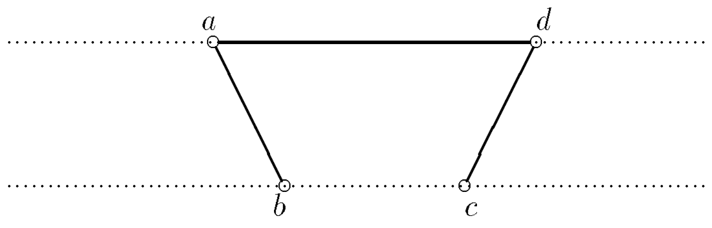


Figure 6.23: An unforbidden order that is not a two-level order. The two horizontal dotted line segments depict the stria of the vertices, and the left-to-right order of the vertices depicts their linear order.

a and d are at most unit distance apart, so are b and c). In fact, the weighted digraph (Definition 5.2) corresponding to this ordered graph has a nonpositive-weight cycle, but it has only positive-weight triangles.

But surely the unordered graph in Figure 6.23 *is* a two-level graph; we need only reorder the vertices, (b, a, d, c) for example, to achieve a two-level realization. Are there non two-level graphs for which the forbidden order in Figure 6.23 arises in all unforbidden orders? Yes, there are; Figure 6.24 shows such a graph. In this graph, one lower neighbour of each upper vertex u must precede u, and one lower neighbour must follow u in any

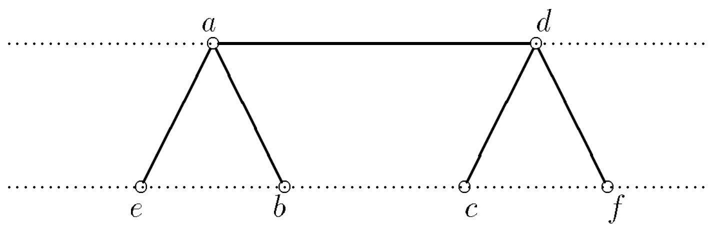


Figure 6.24: The only unforbidden order for this non two-level graph. It contains no forbidden ordered triples.

unforbidden order since the other orders for this triple are indeed forbidden. For any unforbidden order for this graph, label the left upper vertex a, and the right lower neighbour b. Similarly, label the right upper vertex d, and the left lower neighbour c. Clearly, then, Figure 6.23 is an ordered subgraph of any valid ordering of Figure 6.24. It follows that Figure 6.24 is not a two-level graph, yet it cannot be ruled out by any forbidden triple characterization. The following theorem summarizes.

Theorem 6.62 *There is no forbidden-ordered-triple characterization of striated two-level graphs.*

6.5.4 Incomparability Between Different Thicknesses

What happens to the class of two-level τ-strip graphs as the thickness τ of the strip changes? In particular, is there some threshold value for τ such that the class of two-level τ-strip graphs is the same for any τ less than this value? We will see that this is not the case. In fact, we will see that, for any distinct pair of values τ_1 and τ_2, there are two-level τ_1-strip graphs that are not τ_2-strip graphs, and *vice versa.* In fact, we will see that, for any pair of two-level graph classes distinguished by two values for τ, there are graphs in each that are not in the other. This is true even if the two values for τ are arbitrarily small.

It will be more convenient to deal directly with α than with τ. To this end, define α-TWO to be the set of levelled two-level τ-strip graphs where $\tau = \sqrt{1-\alpha^2}$.

Lemma 6.63 *Let α^* be a rational number, and α_1 be a real number such that $1/2 \leq \alpha_1 < \alpha^* < 1$. Then there is an α_1-TWO graph that is not an α-TWO graph for any α satisfying $\alpha^* \leq \alpha$.*

Proof: Let $\alpha^* = n/d$ where n and d are positive integers. Assume without loss of generality that d is even (otherwise multiply n and d by 2). Construct a striated two-level graph $G = (V, E, l)$ as follows. First, let $V = T \cup B$, where

$$\begin{aligned} T &= \{a_1, a_3, \ldots, a_{d-1}\} \text{ and} \\ B &= \{b_0, b_1, \ldots, b_n\} \cup \{a_0, a_2, \ldots, a_d\}. \end{aligned}$$

Furthermore, identify $a_0 = b_0$ and $a_d = b_n$. The striation simply assigns T to the top stria and B to the bottom:

$$l(v) = \begin{cases} 1 & \text{if } v \in B \\ 2 & \text{if } v \in T. \end{cases}$$

Then let $\tau_1 = \sqrt{1-\alpha_1^2}$ and $\{y_1, y_2\} = \{0, \tau_1\}$. Specify the edge set E by describing a realization $f : T \cup B \to \mathbf{R} \times \{0, \tau_1\}$,

$$x_f(v) = \begin{cases} i\alpha^* & \text{if } v = a_i \\ i & \text{if } v = b_i \end{cases}$$

and $y_f(v) = y_{l(v)}$ for every $v \in V$. Note that

$$x_f(a_d) = d\alpha^* = d(n/d) = n = x_f(b_n),$$

as we would expect since $a_d = b_n$. For example, the graph for $\alpha^* = 5/8$ and $\alpha_1 = 1/2$ is shown in Figure 6.25. By construction, $(a_i, a_{i+1}) \notin E$ for all i, since a_i and a_{i+1} are on

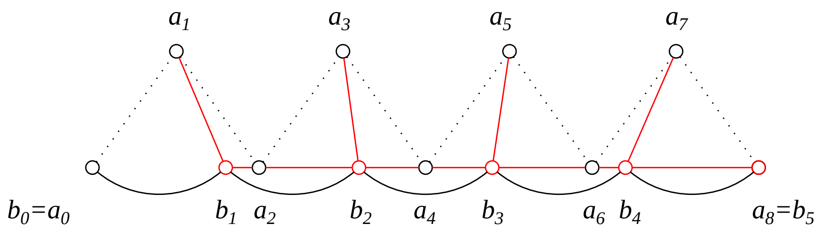


Figure 6.25: A 1/2-TWO graph that is not an α-TWO graph for any $\alpha \geq 5/8$.

different levels so that

$$\begin{aligned} \|f(a_i - a_{i+1})\| &= (\alpha^*)^2 + \tau_1 \\ &> \alpha_1^2 + (1 - \alpha_1^2) \\ &= 1. \end{aligned}$$

Furthermore, $(b_i, b_{i+1}) \in E$ for all i, since $\|f(b_{i+1}) - f(b_i)\| = \|(1,0)\| = 1$. The graph G is connected since $(b_0, b_1, \ldots, b_n)$ is a dominating path, and it is clearly an α_1-TWO graph, since f is its realization.

Now suppose, by way of contradiction, that G is an α-TWO graph for some $\alpha \geq \alpha^*$. Then G has an α-realization where (a_0, a_1) is oriented left to right (otherwise reflect the realization about the y-axis). By Theorem 6.61, the orientations of all nonedges (a_i, a_{i+1}) are determined, in this case left to right since this is consistent with their α_1-realization. Let $D = (V, A, w)$ be the weighted directed graph (Definition 5.16) corresponding to G, l, and the unique (up to reversal) orientation for $\overline{E}$. Then $C = (b_0, b_1, \ldots, b_n = a_d, a_{d-1}, \ldots, a_0 = b_0)$ is a directed cycle in D. Its weight is

$$\begin{aligned} w(C) &= \sum_{i=0}^{n-1} w(b_i, b_{i+1}) + \sum_{i=1}^{d} w(a_i, a_{i-1}) \\ &= n - d\alpha \\ &\leq n - d\alpha^* \\ &= n - d(n/d) = 0. \end{aligned}$$

That is, D has a nonpositive weight cycle, and is therefore not a τ-strip (where $\tau = \sqrt{1-\alpha^2}$) graph by Theorem 5.17. ∎

Lemma 6.64 *Let α^* be a rational number and α_2 be a real number such that $1/2 \leq \alpha^* < \alpha_2 \leq 1$. Then there is an α_2-TWO graph that is not an α-TWO graph for any $\alpha \leq \alpha^*$.*

Proof: This proof is similar to that for Lemma 6.63. Let $\alpha^* = \frac{n}{d}$ where n and d are integers, and d is even, again without loss of generality. Let $G = (V, E)$ where $V = T \cup B$ and

$$\begin{aligned} T &= \{a_1, a_3, \ldots, a_{d-1}\} \text{ and} \\ B &= \{b_0, b_1, \ldots, b_n\} \cup \{a_2, a_4, \ldots, a_d\}. \end{aligned}$$

Again, identify $a_0 = b_0$ and $a_d = b_n$. The striation function is the same: it simply assigns T to the top stria and B to the bottom:

$$l(v) = \begin{cases} 1 & \text{if } v \in B \\ 2 & \text{if } v \in T. \end{cases}$$

Then let $\tau_2 = \sqrt{1-\alpha_2^2}$ and $\{y_1, y_2\} = \{0, \tau_2\}$. Specify the edge set E by describing a realization $f : V \to \mathbf{R} \times \{0, \tau_2\}$.

$$x(v) = \begin{cases} i\alpha_2 & \text{if } v = a_i \\ i\frac{\alpha_2}{\alpha^*} & \text{if } v = b_i \end{cases}$$

and $y_f(v) = y_{l(v)}$ for every $v \in V$. Note that

$$x(b_n) = n\frac{\alpha_2}{\alpha^*} = n\frac{\alpha_2}{(n/d)} = d\alpha_2 = x(a_d),$$

as we would expect since $b_n = a_d$. For example, the graph for $\alpha^* = 5/8$ and $\alpha_2 = 3/4$ is shown in Figure 6.26. By construction, $(a_i, a_{i+1}) \in E$, since a_i and a_{i+1} are on different

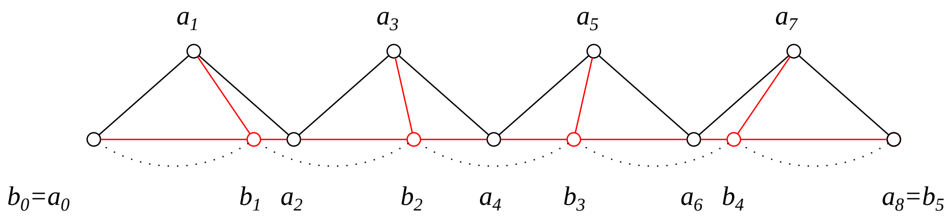


Figure 6.26: A 3/4-TWO graph that is not an α-TWO graph for any $\alpha \leq 5/8$.

levels so that

$$\begin{aligned} \|f(a_i - a_{i+1})\| &= \alpha_2^2 + \tau_2 \\ &= \alpha_2^2 + (1 - \alpha_2^2) \\ &= 1. \end{aligned}$$

Furthermore, $(b_i, b_{i+1}) \notin E$, since $x(b_{i+1}) - x(b_i) = \alpha_2/\alpha^* > 1$ since $\alpha_2 > \alpha^*$. This graph G is connected since $(a_0, a_1, \ldots, a_d)$ is a dominating path, and it is clearly an α_2-TWO graph since f is its realization.

Now suppose, by way of contradiction, that G is an α-TWO graph for some $\alpha \leq \alpha^*$. Then G has an α-realization where (b_0, b_1) is oriented left to right (otherwise reflect the realization about the y-axis). By Theorem 6.61, all nonedges (b_i, b_{i+1}) are determined, and are therefore oriented left to right since this is consistent with their α_2-realization. Let $D = (V, A, w)$ be the weighted directed graph corresponding to G and α. Then $C = (a_0, a_1, \ldots, a_d = b_n, b_{n-1}, \ldots, b_0 = a_0)$ is a directed cycle in D. Its weight is

$$\begin{aligned} w(C) &= \sum_{i=0}^{d-1} w(a_i, a_{i+1}) + \sum_{i=1}^{n} w(b_i, b_{i-1}) \\ &= d\alpha - n \\ &\leq d\alpha^* - n \\ &= d(n/d) - n \\ &= 0. \end{aligned}$$

That is, D has a nonpositive weight cycle, and is therefore not an τ-strip graph by Corollary 5.17.1. ∎

Lemma 6.65 *Let α_1 and α_2 be such that $1/2 \leq \alpha_1 < \alpha_2 \leq 1$. Then α_1-TWO $\not\subseteq$ α_2-TWO and α_2-TWO $\not\subseteq$ α_1-TWO.*

Proof: By the properties of the real numbers, there is a rational α^* such that $\alpha_1 < \alpha^* < \alpha_2$. Then, by Lemma 6.63, there is an α_1-TWO graph that is not an α_2-TWO graph. Similarly, there is an α_2-TWO graph that is not an α_1-TWO graph by Lemma 6.64. ∎

Theorem 6.66 *Let τ_1 and τ_2 satisfy $0 \leq \tau_1 < \tau_2 \leq \sqrt{3}/2$. Then there are two-level τ_1-strip graphs that are not two-level τ_2-strip graphs, and there are two-level τ_2-strip graphs that are not two-level τ_1-strip graphs.*

Proof: By Lemma 6.65, there is a *striated* two-level τ_1-strip graph G that is not a similarly striated two-level τ_2-strip graph. By Property 6.41, this graph G is an induced subgraph of a two-level τ_1-strip graph G', where every realization (for every strip thickness τ) of G' respects the striation for G. Therefore G' is not a two-level τ_2-strip graph, for if it were, its τ_2 realization would also realize the striated graph G. ∎

6.6 Determining the Thickness of the Strip

Let $G = (V, E, s, <)$ be a bistriated (i.e., 2-striated) complement oriented graph. For what values τ is G a striated two-level τ-strip graph? First, let $D = (V, A, w)$ be the weighted digraph (Definition 5.16) corresponding to G and the orientation of its complement. Note that the weights $\pm\alpha$ on the cross arcs are symbolic. Theorem 5.17 tells us that there is a two-level τ-strip realization if every cycle in D has positive weight. What values of τ satisfy this constraint? Let us answer this question by finding the satisfying values for $\alpha = \sqrt{1-\tau^2}$; this is sufficient since $\tau = \sqrt{1-\alpha^2}$.

Since G is bistriated, the weight of any arc in D has one of four values: 1 or -1 for level arcs, and α or $-\alpha$ for cross arcs. Therefore, the weight of any cycle C in D can be expressed as

$$w(C) = L_C + \alpha X_C$$

where L_C and X_C are integers. Note that L_C is the number of level arcs with positive weight, less the number of level arcs with negative weight in C. Similarly, X_C is the number of positive cross arcs, less the number of negative cross arcs. You will find it easy to verify that the weight of a cycle C is positive, that is $L_C + \alpha X_C > 0$, if and only if

$$\begin{aligned} \alpha > \frac{-L_C}{X_C} & \quad \text{if} \quad X_C > 0, \\ L_C > 0 & \quad \text{if} \quad X_C = 0, \quad \text{and} \\ \alpha < \frac{-L_C}{X_C} & \quad \text{if} \quad X_C < 0. \end{aligned} \tag{6.28}$$

We can condense these necessary and sufficient conditions by first defining two parameters r^+ and r^- of the graph as follows.

$$\begin{aligned} r^+ &= \max\left(\left\{\frac{-L_C}{X_C} : X_C > 0\right\} \cup \{-|V|\}\right) \\ r^- &= \min\left(\left\{\frac{-L_C}{X_C} : X_C < 0\right\} \cup \{|V|\}\right). \end{aligned}$$

Note that, if $X_C \neq 0$, then $|X_C| \geq 1$ and $|L_C| < |V|$, so $|-L_C/X_C| < |V|$, since $|X_C| + |L_C| \leq |C| \leq |V|$. Therefore $r^+ = -|V|$ if and only if $X_C \leq 0$ for all cycles C. Similarly, $r^- = |V|$ if and only if $X_C \geq 0$ for all cycles C. We have established the following theorem.

Theorem 6.67 *Let $G = (V, E, l, <)$ be a bistriated, complement oriented graph, and let $D = (V, A, w)$ be the corresponding weighted digraph. Then G is a two-level τ-strip graph if and only if*

1. *$r^+ < \sqrt{1-\tau^2} < r^-$, and*

2. *$L_C > 0$ for all cycles C in D for which $X_C = 0$,*

where $0 \le \tau \le \sqrt{3}/2$.

An easy corollary is that two-level graphs appear in "bands" as we sweep through values for τ.

Corollary 6.67.1 *Let $G = (V, E, l, <)$ be both a two-level τ_1-strip graph and a two-level τ_2-strip graph. Then G is a two-level τ-strip graph for all τ in between.*

Proof: Assume without loss of generality that $\tau_1 \le \tau_2$, and let τ be some value between τ_1 and τ_2. Then $\sqrt{1-\tau_2^2} \le \sqrt{1-\tau^2} \le \sqrt{1-\tau_1^2}$. By the theorem,

1. $r^+ < \sqrt{1-\tau_2^2} \le \sqrt{1-\tau^2} \le \sqrt{1-\tau_1^2} < r^-$, and

2. $L_C > 0$ for all cycles C in D for which $X_C = 0$.

Again, by the theorem, G is a two-level τ-strip graph. ∎

Another corollary allows us to tell if a graph is a two-level graph for some strip thickness.

Corollary 6.67.2 *Let $G = (V, E, l, <)$ be a bistriated, complement oriented graph and let $D = (V, A, w)$ be the corresponding weighted digraph. Then G is a bistriated, complement oriented, two-level τ-strip graph for some τ if and only if (1) the intervals $[1/2, 1]$ and (r^+, r^-) intersect, and (2) $L_C > 0$ for all cycles C in D for which $X_C = 0$.*

A Recognition Algorithm for Bistriated, Complement Oriented, Two-Level Graphs

We can use Theorem 6.67 to recognize bistriated two-level graphs. Essentially, we just perform a binary search on the interval $[1/2, 1]$ for an α value. The value $\tau = \sqrt{1-\alpha^2}$

will turn our striated graph into a levelled graph, and we can then apply an existing algorithm to recognize it. If the graph is not a levelled τ-strip graph for this value of τ, then the corresponding directed graph will have a nonpositive cycle, which we can then analyze to determine whether α should be larger or smaller. The complete algorithm appears in Table 6.6.

Table 6.6: **Algorithm:** 2STRIAE(G) [Recognize striated two-level graphs]

Input: A bistriated, complement oriented graph $G = (V, E, s, <)$
Output: A realization $f : V \to \mathbf{R} \times \{0, \tau\}$ showing that G is a bistriated, complement oriented, τ-strip graph, or
a message stating that G is not a two-level graph

1 $r^+ \leftarrow 1/2$
2 $r^- \leftarrow 1$
3 **while** $r^+ < r^-$
4   **do** $\alpha \leftarrow (r^+ + r^-)/2$
5         $\tau \leftarrow \sqrt{1 - \alpha^2}$
6   Define $l' : V \to [0, \tau]$, where $l'(v) = (0$ if $s(v) = 1$; τ if $s(v) = 2)$
7   Define $G' = (V, E, l', <)$
8         $(C, f) \leftarrow$ STRIP-LAYOUT(G') /* From Table 5.2 */
9   **if** $w(C) > 0$
10   **then return** f
11   **halt**
12   **else** compute L_C and X_C from cycle C
13   **if** $X_C = 0$
14   **then return** "G is not a striated two-level graph"
15   **halt**
16   **else if** $X_C < 0$
17   **then** $r^+ \leftarrow -L_C/X_C$
18   **else** $r^- \leftarrow -L_C/X_C$
20 **return** "G is not a striated two-level graph"

Steps 18 and 19 would split the interval $[r^+, r^-]$ exactly in half if they set the limits to the midpoint α. In fact, these steps do at least as well by setting the limits to

$-L_C/X_C$. Suppose $X_C < 0$. Then by Equation 6.28, if $\alpha > -L_C/X_C$, then C would be positive. But it is not, so $\alpha \leq -L_C/X_C$. Is it safe to increase r^+ this much? Yes, it is, since Theorem 6.67 states that α must be greater than the maximum of all such ratios. Similarly, if $X_C > 0$, then $-L_C/X_C \leq \alpha$, and it is safe and efficacious to set r^- to this ratio, thereby reducing the interval $[r^+, r^-]$ by at least one half.

How can we analyze the run time of this algorithm? Let us begin by noting that L_C and X_C are integers, that $-|V| \leq L_C \leq |V|$, and that $-|V| \leq X_C \leq |V|$. Therefore, the ratio $-L_C/X_C$ can take on at most $O(V^2)$ values and, in particular, at most $O(V^2)$ values in the interval $[0, 1]$. Since the search interval is reduced by at least half on each iteration, the algorithm halts in $O(\log V^2) = O(\log V)$ time. From Theorem 5.17, we know that Step 8 takes $O(V^3)$ time. We have established the following theorem.

Theorem 6.68 *Algorithm 2STRIAE (Table 6.6) recognizes bistriated, complement oriented strip graphs in $O(V^3 \log V)$ time.*

Chapter 7

Conclusion

We have seen that the recognition problem for unit disk graphs is **NP**-hard. Furthermore, several familiar **NP**-complete problems on graphs remain **NP**-complete for unit disk graphs even if the graphs are represented by their realizations as intersecting disks. Nevertheless, at least one nontrivial problem on unit disk graphs, namely the maximum clique problem, is solvable in polynomial time if a realization is available.

We have also seen that one-dimensional unit disk graphs, in which the centres of the realizing disks are constrained to be collinear, are indifference graphs. What effect does the second dimension have on the complexity of graph theoretic problems? The classes of τ-strip graphs model a gradual introduction of this second dimension.

Indifference graphs are equivalently 0-strip graphs or 1-level graphs. These graphs are easy to recognize, and have very efficient algorithms for the kinds of problems discussed in this thesis. The key to their tractability seems to be their characterization as graphs with a linear order on their vertices such that for any three vertices $u < v < w$, if u and w are adjacent, then v is adjacent to *both* u and w (Theorem 2.3.11).

Indifference graphs are also cocomparability graphs, as are all τ-strip graphs from $\tau = 0$ (indifference graphs) to $\tau = \sqrt{3}/2$ (but not with $\tau > \sqrt{3}/2$). We saw in Chapter 4 that such graphs also admit efficient algorithms, though the algorithms for indifference graphs are even more efficient. Again, the key seems to be that a linear order (the spanning order) can be imposed on their vertices, and that such an order is easy to find (Lemma 4.2). Recall that a spanning order is a linear order on the vertices of a graph such that, for any three vertices $u < v < w$, if u and w are adjacent, then v is adjacent to *either* u or w (or both) (Definition 4.1).

The spanning order is not the only constraint on τ-strip graphs. Chapter 5 captures some additional constraints by characterizing strip graphs in terms of cycles in a weighted digraph. This characterization allows us to verify that some small graphs are not strip graphs. Furthermore, efficient algorithms could exploit the implicit representation of edges in a strip graph realization. This exploitation, set up in Chapter 3, actually applies to arbitrary unit disk graphs, but we saw in Chapter 5 how to combine this representation with algorithms for cocomparability graphs to improve on both.

Strip graphs that have two-level realizations—where the disks lie on the boundary of the strip—have an added combinatorial nature to them that also can be exploited. Chapter 6 illustrates this with its transitive reduction and minimum weight maximal clique algorithms on the oriented complement of a two-level graph, and its weighted independent dominating set algorithm on two-level graphs. Two-level graphs also severely restrict the realizations of the claw and the square, where they appear as induced subgraphs. This restriction ensures that every realization has essentially the same left-to-right order among its distal vertices. Such levelled, oriented two-level graphs can be recognized in polynomial time.

We exhibited several connections between two-level graphs and other perfect graphs. Recall that a *level-edge graph* has a realization like a two-level graph, but only vertices on the same level may be adjacent. Similarly, a *cross-edge graph* has a realization like a two-level graph, but only vertices on different levels may be adjacent. Clearly, every two-level graph is the edge-disjoint union of a level-edge graph and a cross-edge graph. The class-inclusion diagram shown in Figure 7.1 summarizes the place of unit disk graphs and relatives presented in Chapter 6. The class labelled "strip" is itself a hierarchical family of τ-strip graph classes, one for each value of τ between 0 and $\sqrt{3}/2$. The class labelled "2-level" is also a family of classes, one for each value of τ between 0 and $\sqrt{3}/2$, but no two subclasses are comparable.

7.1 Epilogue

Alice presented her work to the requirements team at Blue Sky Airlines, and outlined her graph theoretic model of the midnight bell problem. She carefully explained unit disk graphs, strip graphs, two-level graphs, and cocomparability graphs. She described her efficient solution to the dominating set problem on cocomparability graphs, and provided context by describing other domination problems.

The team thanked Alice for her presentation, and agreed that her graph theoretic model is appropriate. For them, however, the graph theory highlighted some inadequacies of their requirements. In particular, they realized that it is unacceptable for an airplane to both send a radio beacon signal and to receive another one; the two signals would interfere. They felt that their problem would be better modelled by the independent dominating set problem (§4.2.4) in which no two transmitters can be within range of one another. This made Alice very happy because she had a better solution for this problem (Theorem 4.35) than for the unconstrained dominating set problem (Theorem 4.18).

Furthermore, the specifications team agreed that Blue Sky's airplanes would follow the "rules-of-the-road" and stay to the edges of the corridor. Alice could hardly believe her luck, since she knew how to solve this problem (on two-level graphs) *very* well (Corollary 6.23.1), without having to do any additional research (Alice likes research, but she also likes to see working solutions). Alice rushed off to implement her solution, already excited about more opportunities to exploit geometric constraints in graph theory.

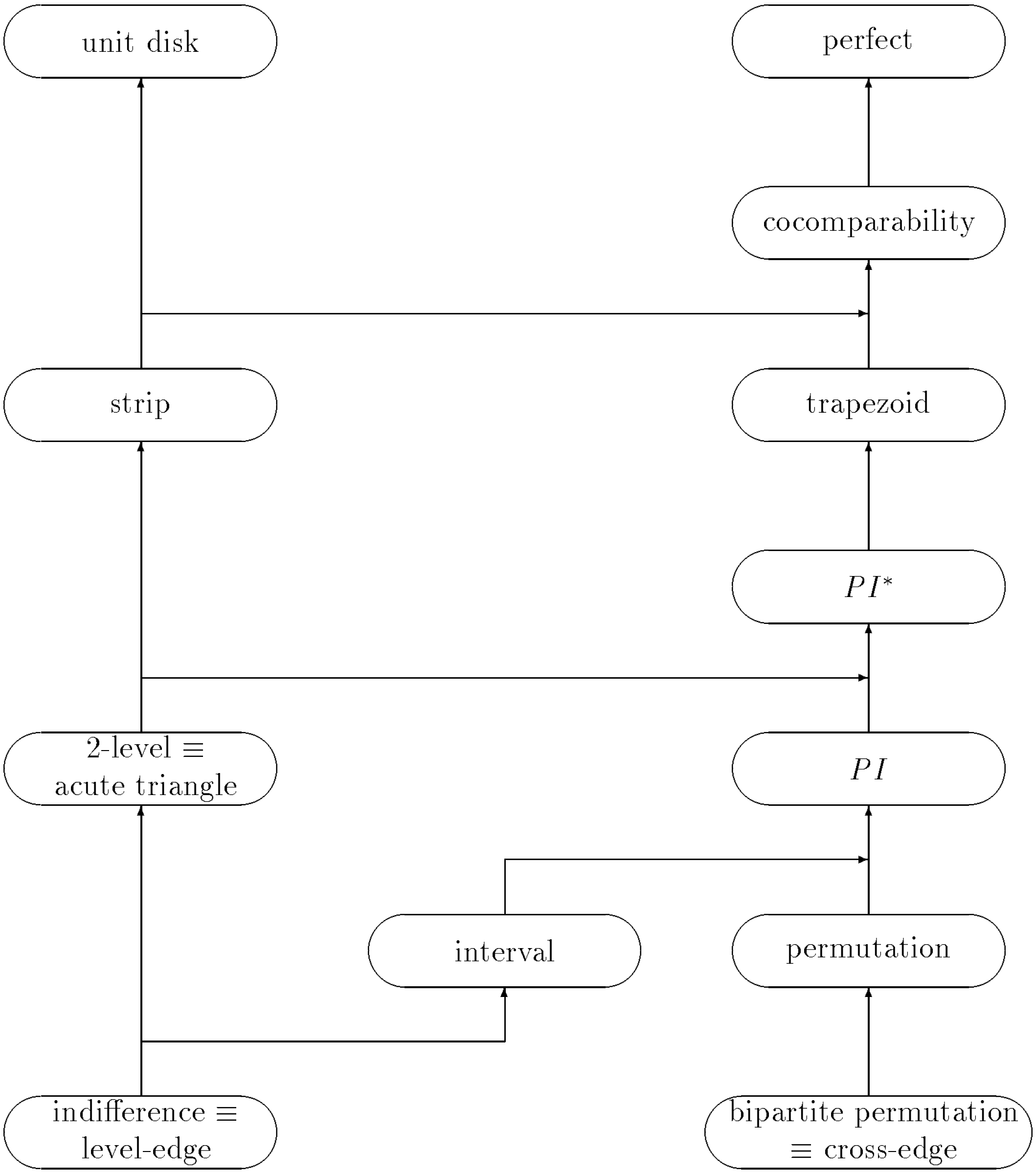


Figure 7.1: A Hierarchy of Graphs. The relation "class1 → class2" means that class1 graphs are a proper subclass of class2 graphs.

Bibliography

[AESW90] Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, pages 203–210, Berkeley, California, June 6–8 1990. ACM, ACM Press.

[AGU72] A.V. Aho, M.R. Garey, and J.D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[AH85] David Avis and Joe Horton. Remarks on the sphere of influence graph. In J.E. Goodman, E. Lutwak, J. Malkevitch, and R. Pollack, editors, *Discrete Geometry and Convexity*, pages 323–327. The New York Academy of Sciences, 1985. Annals of the New York Academy of Sciences, Vol. 440.

[AH76] K.I. Appel and W. Haken. Every planar map is four colorable. *Bull. Am. Math. Soc.*, 82:711–712, 1976. Cited by [SK86].

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[AIKS89] Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. In *Proceedings of the Fifth Annual Symposium on Computational Geometry*, pages 283–291, Saarbrücken, West Germany, June 5–7 1989. ACM Press.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 14–23, Pittsburgh, PA, October 24–27 1992. IEEE Computer Society Press.

[And88] Thomas Andreae. On the unit interval number of a graph. *Discrete Applied Mathematics*, 22:1–7, 1988.

[AR92] K. Arvind and C. Pandu Rangan. Connected domination and Steiner set on weighted permutation graphs. *Information Processing Letters*, 41:215–220, 1992.

[BB86] Alan A. Bertossi and Maurizio A. Bonuccelli. Hamiltonian ciruits in interval graph generalizations. *Information Processing Letters*, 23:195–200, 1986.

[BC87] Sandeep N. Bhatt and Stavros S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Information Processing Letters*, 25(4):263–267, 1987.

[Bel58] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958. Cited by [CLR90].

[Ber61] Claude Berge. Farbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind (Zusammenfassung). *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg, Math.-Natur, Reihe*, 10:114–115, 1961. Cited by [Gol80].

[Ber86] Alan A. Bertossi. Total domination in interval graphs. *Information Processing Letters*, 23:131–134, 1986.

[BK87] Andreas Brandstädt and Dieter Kratsch. On domination problems for permutation and other graphs. *Theoretical Computer Science*, 54:181–198, 1987.

[BK93] Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is **NP**-hard. Technical Report 93-27, University of British Columbia, August 1993.

[BL76] K.S. Booth and G.S. Luecker. Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.

[BLW76] Norman L. Biggs, E. Keith Lloyd, and Robin J. Wilson. *Graph Theory 1736–1936.* Oxford University Press, 1976.

[BSS87] Andreas Brandstädt, Jeremy Spinrad, and Lorna Stewart. Bipartite permutation graphs are bipartite tolerance graphs. *Congressus Numerantium*, 58:165–174, 1987.

[BSW77] Jon L. Bentley, Donald F. Stanat, and E. Hollins Williams, Jr. The complexity of finding fixed-radius near neighbors. *Information Processing Letters*, 6(6):209–212, December 1977.

[Bur82] Stephan A. Burr. An **NP**-complete problem in Euclidean Ramsey theory. *Congressus Numerantium*, 35(4):131–138, 1982.

[Can88] John Canny. Some algebraic and geometric computations in **PSPACE**. In *Proceedings of the 20th Annual Symposium on the Theory of Computing*, pages 460–467, Chicago, Illinois, 2–4 May 1988.

[CCJ90] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, pages 165–177, 1990.

[Cha85] Bernard Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, IT-31(4):509–517, July 1985. Cited by [HS89].

[Cha94] Maw Shang Chang. Weighted domination on cocomparability graphs. Unpublished manuscript, 1994.

[Che90] Faithful H.K. Cheah. *A Recognition Algorithm for II-graph.* PhD thesis, University of Toronto, Department of Computer Science, December 1990. Technical Report 246/90.

[CK87] D.G. Corneil and P.A. Kamula. Extensions of permutation and interval graphs. *Congressus Numerantium*, 58:267–275, 1987.

[CLR90] Thomas J. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 1990.

[CM78] Michael Capobianco and John C. Molluzzo. *Examples and Counterexamples in Graph Theory.* Elsevier North-Holland, Inc., 1978.

[Coo71] Steven A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual Symposium on Theory of Computing*, pages 151–158, New York, 1971. Association for Computing Machinery. Cited by [GJ79].

[Coz92] Margaret Barry Cozzens, May 1992. Private communication.

[CS88] John Horton Conway and Neil James Sloane. *Sphere Packings, Lattices and Groups.* Number 290 in Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1988.

[CS90a] Charles J. Colbourn and Lorna K. Stewart. Permutation graphs: Connected domination and Steiner trees. *Discrete Mathematics*, 86:179–189, 1990.

[CS90b] Derek G. Corneil and Lorna K. Stewart. Dominating sets in perfect graphs. *Discrete Mathematics*, 86:145–164, 1990.

[CW87] Don Coppersmith and Shmuel Winograd. Matrix multiplications via arithmetic progressions. In *Proceedings of the Ninetheenth Annual ACM Symposium on Theory of Computing*, pages 1–6, New York, 1987. Association for Computing Machinery. Cited by [CLR90].

[Dam90] Peter Damaschke. Forbidden ordered subgraphs. In R. Bodendiek and R. Henn, editors, *Topics in Combinatorics and Graph Theory*, pages 219–229. Physica-Verlag, Heidelberg, 1990.

[Dam92] Peter Damaschke. Distances in cocomparability graphs and their powers. *Discrete Applied Mathematics*, 35:67–72, 1992.

[DD90] Matthew T. Dickerson and R. Scot Drysdale. Fixed radius search problem for points and segments. *Information Processing Letters*, 35(5):269–273, 1990.

[Del34] B. Delaunay. Sur la sphère vide. *Bull. Acad. Sci. USSR(VII)*, pages 793–800, 1934. Cited by [PS85].

[DGP88] Ido Dagan, Martin Charles Golumbic, and Ron Yair Pinter. Trapezoid graphs and their coloring. *Discrete Applied Mathematics*, 21:35–46, 1988.

[DGS84] U. Derigs, O. Goetke, and R. Schrader. Bisimplicial edges, Gaussian elimination and matchings in bipartite graphs. Technical Report 84332-OR, Universitat Bonn, West Germany, 1984. Cited by [BSS87].

[DH73] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.

[DM41] Ben Dushnik and E.W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63:600–610, 1941. Cited by [Gol80].

[DS94] Jitender S. Deogun and George Steiner. Polynomial algorithms for hamiltonian cycle in cocomparability graphs. *SIAM Journal on Computing*, 23(3):520–552, June 1994.

[Duc79] Pierre Duchet. *Representations, Noyaux en Theorie des Graphes et Hypergraphes.* PhD thesis, University of Paris 6, 1979. Cited by [Dam90].

[Duc84] Pierre Duchet. Classical perfect graphs. *Annals of Discrete Mathematics*, 21:67–96, 1984.

[Epp95] Susanna S. Epp. *Discrete Mathematics with Applications—Second Edition.* PWS Publishing Company, 1995.

[EW94] Peter Eades and Sue Whitesides. The realization problem for Euclidean minimum spanning trees is **NP**-hard. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 49–56, Stony Brook, NY, 1994. ACM, ACM Press.

[FG65] D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–355, 1965.

[Fis83] Peter C. Fishburn. On the sphericity and cubicity of graphs. *Journal of Combinatorial Theory, Series B*, 35:309–318, 1983.

[FK85] Martin Farber and J. Mark Keil. Domination in permutation graphs. *Journal of Algorithms*, 6:309–321, 1985.

[Gal67] T. Gallai. Transitiv orientierbare Graphen. *Acta Math. Acad. Sci. Hung.*, 18:25–66, 1967. Cited by L. Stewart, personal communication, March 1996.

[GH64] P.C. Gilmore and A.J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canad. J. Math.*, 16:539–548, 1964.

[GHS89] C.P. Gabor, W.L. Hsu, and K.J. Supowit. Recognizing circle graphs in polynomial time. *Journal of the ACM*, 36:435–473, 1989. Cited by [van90].

[GJ77] M.R. Garey and D.S. Johnson. The rectilinear Steiner tree problem is **NP**-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, June 1977.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: a Guide to the Theory of* **NP**-*completeness*. W.H. Freeman and Company, 1979.

[GLL82] U.I. Gupta, D.T. Lee, and J.Y.-T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12(4):459–467, Winter 1982.

[GLS84] M. Grötschel, Laslo Lovász, and A. Schrijver. Polynomial algorithms for perfect graphs. *Annals of Discrete Mathematics*, 21:325–356, 1984.

[GM82] Martin C. Golumbic and Clyde L. Monma. A generalization of interval graphs with tolerances. *Congressus Numerantium*, 35(4):321–331, 1982.

[God88] Erhard Godehardt. *Graphs as Structural Models: the Application of Graphs and Multigraphs in Cluster Analysis*, volume 4 of *Advances in system analysis*. Vieweg, 1988.

[Gol80] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

[Grä95] Albert Gräf. *Coloring and Recognizing Special Graph Classes*. PhD thesis, Musikwissenschaftliches Institut, Abteilung Musikinformatik, Johannes Gutenberg-Universität Mainz, February 1995. Available as technical report Bericht Nr. 20.

[Gre89] Angelo Gregori. Unit-length embedding of binary trees on a square grid. *Information Processing Letters*, 31:167–173, 1989.

[GRU83] Martin C. Golumbic, Doron Rotem, and Jorge Urrutia. Comparability graphs and intersection graphs. *Discrete Mathematics*, 43:37–46, 1983.

[GS69] K.R. Gabriel and R.R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.

[Had77] F.O. Hadlock. A shortest path algorithm for grid graphs. *Networks*, 7:323–334, 1977.

[Hal80] William K. Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514, 1980.

[Hal90] M.M. Halldórsson. A still better performance guarantee for approximate graph coloring. Technical Report 91-35, DIMACS, New Brunswick, N.J., 1990. Cited by [JT95].

[Hav82a] Timothy Franklin Havel. *The Combinatorial Distance Geometry Approach to the Calculation of Molecular Conformation.* PhD thesis, University of California, Berkeley, 1982. Cited by [Fis83].

[Hav82b] Timothy Franklin Havel. The combinatorial distance geometry approach to the calculation of molecular conformation. *Congressus Numerantium*, 35(4):361–371, 1982.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. Cited by [PS82].

[HKC83] Timothy Franklin Havel, Irwin D. Kuntz, and Gordon M. Crippen. The combinatorial distance geometry method for the calculation of molecular conformation. I. A new approach to an old problem. *Journal of Theoretical Biology*, 35(4):359–381, 1983.

[HS89] John Hershberger and Subhash Suri. Finding tailored partitions. In *Proceedings of the Fifth Annual Symposium on Computational Geometry*, pages 255–265, Saarbrücken, West Germany, June 5–7 1989. ACM Press.

[IA83] Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 4:310–323, 1983.

[IA86] Hiroshi Imai and Takeo Asano. Efficient algorithms for geometric graph search problems. *SIAM Journal on Computing*, 15(2):478–494, 1986.

[Ima82] Hiroshi Imai. Finding connected components of an intersection graph of squares in the Euclidean plane. *Information Processing Letters*, 15(3):125–128, 1982.

[IPS82] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal of Computing*, 11(4):676–686, 1982.

[Irv91] Robert W. Irving. On approximating the minimum independent dominating set. *Information Processing Letters*, 37:197–200, 28 February 1991.

[Jac92] Zygmunt Jackowski. A new characterization of proper interval graphs. *Discrete Mathematics*, 105:103–109, 1992.

[JAMS91] David S. Johnson, Cecilia A. Aragon, Lyle.A. McGeogh, and Catherine Schevon. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partioning. *Operations Research*, 39(3):378–406, May-June 1991.

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Science*, 9:256–278, 1974. Cited by [MBH+95].

[JT95] Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, 1995.

[Kas80] T. Kashiwabara. Algorithms for some intersection graphs. In *Graph Theory and Algorithms*, Lecture Notes in Computer Science Vol. 108, pages 171–181, Sendai, Japan, October 24-25 1980. Springer-Verlag.

[Kei85] J. Mark Keil. Finding hamiltonian circuits in interval graphs. *Information Processing Letters*, 20:201–206, 1985.

[Kei86] J. Mark Keil. Total domination in interval graphs. *Information Processing Letters*, 22:171–174, 1986.

[KM94] Jan Kratochvíl and Jiří Matoušek. Intersection graphs of segments. *Journal of Combinatorial Theory, Series B*, 62:289–315, November 1994.

[Kra91a] Jan Kratochvíl. String graphs. I. The number of critical nonstring graphs is infinite. *Journal of Combinatorial Theory, Series B*, 52(1):53–66, May 1991.

[Kra91b] Jan Kratochvíl. String graphs. II. Recognizing string graphs is **NP**-hard. *Journal of Combinatorial Theory, Series B*, 52(1):67–78, 1991.

[Kra94] Jan Kratochvíl. A special planar satisfiability problem and a consequence of its **NP**-completeness. *Discrete Applied Mathematics*, 52:233–252, 1994.

[KS93] Dieter Kratsch and Lorna Stewart. Domination on cocomparability graphs. *SIAM J. Disc. Math.*, 6(3):400–417, 1993.

[Kur30] Kazimierz Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930. Cited by [BLW76].

[LB62] C.G. Lekkerkerker and J.Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51:45–64, 1962.

[Lia94] Y. Daniel Liang. Some efficient algorithms in trapezoid graphs and cocomparability graphs. Unpublished manuscript, 2 March 1994.

[LO93] Peter J. Looges and Stephan Olariu. Optimal greedy algorithms for indifference graphs. *Computers Math. Applic.*, 25(7):15–25, 1993.

[Lov72] Lásló Lovaász. A characterization of perfect graphs. *Journal of Conbinatorial Theory, Series B*, 13:95–98, 1972. Cited by [Gol80].

[LS92] Hans-Peter Lenhof and Michiel Smid. Enumerating the k closest pairs optimally. In *33rd Annual Symposium on Foundations of Computer Science*, pages 380–386, Pittsburgh, Pennsylvania, October 24–27 1992. IEEE Computer Society Press.

[LS94] Hans-Peter Lenhof and Michiel Smid. An animation of a fixed-radius all-nearest neighbors algorithm. In Marc H. Brown and John Hershberger, editors, *The Third Annual Video Review of Computational Geometry*, chapter 3, pages 9–10. Digital Equipment Corporation, 30 December 1994. SRC Research Report 133a & 133b.

[LY93] C. Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. In *Proceedings of the 25th Annual Symposium on the Theory of Computing*, pages 286–293. ACM, 1993. Cited by [JT95].

[Mae84] Hiroshi Maehara. Space graphs and sphericity. *Discrete Applied Mathematics*, 7:55–64, 1984.

[Mae86] Hiroshi Maehara. Sphericity exceeds cubicity for almost all complete bipartite graphs. *Journal of Combinatorial Theory, Series B*, 40:231–235, 1986.

[MB83] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.

[MBH+95] M.V. Marathe, H. Breu, H.B. Hunt II, S.S. Ravi, and D.J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.

[MHR92] M.V. Marathe, H.B. Hunt II, and S.S. Ravi. Geometry based approximations for intersection graphs. In *Proceedings of the Fourth Canadian Conference on Computational Geometry*, pages 244–249, 1992.

[MS80] D.W. Matula and R.R. Sokal. Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis*, 12:205–222, July 1980.

[MS91] Tze-Heng Ma and Jeremy Spinrad. An $O(n^2)$ time algorithm for the 2-chain cover problem and related problems. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 363–372, San Francisco, California, January 28–30 1991.

[Möh85] Rolf H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In Ivan Rival, editor, *Graphs and Orders: the Role of Graphs in the Theory of Ordered Sets and Its Applications*, pages 41–101. D. Reidel Publishing Company, New York and London, May 18–31, 1984 1985. Proceedings of the NATO Advanced Study Institute on Graphs and Order.

[NC88] T. Nishizeki and N. Chiba. *Planar Graphs: Theory and Algorithms*, volume 32 of *Annals of Discrete Mathematics*. North-Holland, 1988.

[OvL81] M. Overmars and H. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23:166–204, 1981.

[Pee91] R. Peeters. On coloring j-unit sphere graphs. Technical Report FEW 512, Tilburg University, NL, 1991. Cited by [Grä95].

[Phi90] Thomas K. Philips. New algorithms to color graphs and find maximum cliques. report RC 16326 (#72348), IBM Research Division, 16 November 1990.

[PLE71] Amir Pnueli, Abraham Lempel, and Shimon Even. Transitive orientation of graphs and identification of permutation graphs. *Canadian Journal of Mathematics*, 23:160–175, 1971. Cited by [Gol80].

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, 1985.

[Rob68a] Fred S. Roberts. Indifference graphs. In *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, New York and London, February 1968. Proceedings of the Second Ann Arbor Graph Theory Conference.

[Rob68b] Fred S. Roberts. On the boxicity and cubicity of a graph. In *Recent Progress in Combinatorics*, pages 301–310. Academic Press, New York and London,

1968. Proceedings of the Third Waterloo Conference on Combinatorics, May 1968.

[Rob78] Fred S. Roberts. *Graph Theory and Its Applications to Problems of Society*, volume 29 of *Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, 1978.

[RR88a] G. Ramalingam and C. Pandu Rangan. Total domination in interval graphs revisted. *Information Processing Letters*, 27:17–21, 1988.

[RR88b] G. Ramalingam and C. Pandu Rangan. A unified approach to domination problems on interval graphs. *Information Processing Letters*, 27:271–274, 1988.

[Sac94] Horst Sachs. Coin graphs, polyhedra, and conformal mapping. *Discrete Mathematics*, 134:133–138, 1994.

[SBS87] Jeremy Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18:279–292, 1987.

[Sch85] Edward R. Scheinerman. Characterizing intersection classes of graphs. *Discrete Mathematics*, 55(2):185–193, July 1985.

[Sed83] Robert Sedgewick. *Algorithms*. Addison-Wesley Publishing Company, 1983.

[SH75] M.I. Shamos and D.J. Hooey. Closest-point problems. In *16th Annual Symposium of Foundations of Computer Science*, pages 151–162. IEEE, 1975.

[Sim88] Klaus Simon. An improved algorithm for transitive closure on acyclic digraphs. *Theoretical Computer Science*, 58:325–346, 1988.

[SK86] Thomas L. Saaty and Paul C. Kainen. *The Four-Color Problem: Assaults and Conquest*. Dover, 1986.

[Spi85] Jeremy Spinrad. On comparability and permutation graphs. *SIAM Journal on Computing*, 14(3):658–670, August 1985.

[Spi94] Jeremy Spinrad. Dimension and algorithms. In Vincent Bouchité and Michel Morvan, editors, *Orders, Algorithms, and Applications*, Lecture Notes in Computer Science 831, pages 33–52, Lyon, France, 4–8 July 1994. Springer-Verlag. Proc. International Workshop ORDAL '94.

[Sup83] K.J. Supowit. The relative neighborhood graph with an application to minimum spanning trees. *Journal of the ACM*, 30(3):428–447, July 1983.

[Tar83] Robert Endre Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[TH90] K.H. Tsai and W.L. Hsu. Fast algorithms for the dominating set problem on permutation graphs. In Asano et al., editor, *SIGAL '90 Algorithms*, Lecture Notes in Computer Science, pages 109–117. Springer-Verlag, 1990.

[TJS76] William T. Trotter, Jr., John I. Moore, Jr., and David P. Sumner. The dimension of a comparability graph. *Proc. Amer. Math. Soc.*, 60:35–38, 1976. Cited by [Gol80].

[Tou80] G.T. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.

[Tuc80] Alan C. Tucker. An efficient test for circular-arc graphs. *SIAM Journal on Computing*, 9:1–24, 1980. Cited by [GLL82].

[Tur91] Volker Turau. Fixed-radius near neighbors search. *Information Processing Letters*, 39:201–203, 1991.

[Val81] Leslie G. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, C-30(2):135–140, February 1981.

[van90] Jan van Leeuwen. Graph algorithms. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, pages 527–631. Elsevier and the MIT Press, 1990.

[Vor08] G. Voronoi. Nouvelles applications des parametres continus à la theorie des formes quadratiques. deuxième mémorie: Recherches sur les parallélоèdres primitifs. *J. reine angew. Math.*, 134:198–287, 1908. Cited by [PS85].

[Weg67] G. Wegner. *Eigenschaften der Nerven Homologish-Einfacher Familien im R^m*. PhD thesis, Göttingen, 1967. Cited by [CM78].

[Yan82] Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM J. Alg. Disc. Meth.*, 3(3):351–358, September 1982.

[Yao82] Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

[Zah71] Charles T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, C-20(1):68–86, January 1971.

Appendix A

A Data Structure for Maintaining "Test Tube" Maxima

This appendix defines the "test tube" maxima of a set S of points, describes their properties, and shows how to maintain a set of them in a data structure. It describes an algorithm that builds the data structure in $O(S \log S)$ time with $O(S)$ space. This data structure then supports binary search[1] in $O(\log S)$ time and deletion in $O(\log S)$ amortized time.

We adapt Hershberger and Suri's circular hull data structure [HS89] for our purposes. They in turn adapt the dynamic convex hull data structures due to Overmars and van Leeuwen [OvL81] and Chazelle [Cha85]. Although some details are identical, all are presented here to make this appendix self contained.

A.1 Definitions

The *test tube* (or just *tube*) centered at (x, y) is the union of the closed unit radius disk centered at (x, y) and the infinite rectangle $[x-1, x+1] \times [y, \infty)$. Let T_l and T_r be two tubes centered at (x_l, y_l) and (x_r, y_r) respectively. Then T_l *lies to the left* of T_r, and T_r *lies to the right* of T_l, if $x_l < x_r$. Tube T_l *lies strictly to the left* of T_r, and T_r *lies strictly to the right* of T_r, if $x_l + 1 < x_r - 1$.

Let the *sites* be a finite set S of points in the plane. For simplicity of presentation, assume that S has at most unit diameter, that is, that every pair of sites is within unit

[1] For example, Section 3.2.2 uses binary search to find a maximal point in a query test tube.

distance,[2] and that no two points in S have the same x-coordinate. However, the material in this appendix is easily extended to sites that do not satisfy these assumptions.

A tube T *contains* a subset $P \subseteq S$ of sites if $P \subseteq T$, and it is *empty* if it does not contain any sites. A tube and a site on its boundary are said to be *incident*. A tube is said to be *supporting (with respect to S)* if it is incident to exactly one site and is otherwise empty. A site in S is said to be a *test tube maximal site with respect to* S, or just a *maximal*, if it is incident to some supporting tube. Let ∂S denote the set of maxima with respect to S. Let the sites, and therefore also their maxima, be ordered by nondecreasing x-coordinate, that is, from left to right. Two sites are *neighbouring maxima* if they are maximal and adjacent in left to right order. The *left neighbour* of a maximal p is the maximal immediately before p in the order. Similarly, p's *right neighbour* is the maximal immediately after p. The *tube* T_{pq} *through (distinct sites)* p *and* q is the unique tube incident to both p and q. Note that T_{pq} is always well defined since $|x_p - x_q| \leq 1$ for all pairs p and q of sites.

A.2 Test Tubes, Maxima, and Their Properties

The arguments in this appendix make extensive use of three basic operations on tubes: lowering, raising, and rotating. To *lower* a tube, continuously translate its center to a smaller y-coordinate. Note that if tube T' is a lowered version of T, then $T \subset T'$. Similarly, *raise* a tube by continuously translating its center to a greater y-coordinate. Again, note that if tube T' is a raised version of T, then $T' \subset T$. Finally, to *rotate* a tube about an incident site p, continuously rotate the center of the tube about p while keeping the tube "upright" and the site incident, as shown in Figure A.1.

The next few lemmas clarify the behaviour of test tube maxima. The first lemma

[2] This *is* the case for the application of this material in Section 3.2.2.

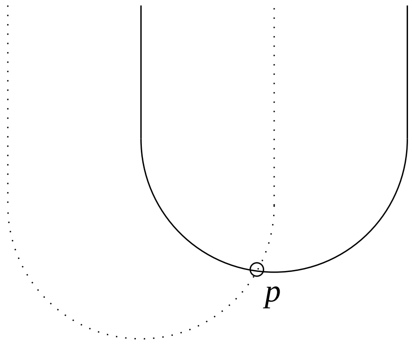


Figure A.1: A test tube rotating clockwise about site p.

elucidates the behaviour of a rotating test tube. The subsequent lemma is straightforward and justifies the term "maximal".

Lemma A.1 (The Rotation Lemma) *As a test tube rotates clockwise about a point p, it can only gain sites from the right of p, and it can only lose sites to the left of p. Similarly, a counterclockwise rotating test tube gains only from the left and loses only to the right.*

Proof: At any instant, p partitions the boundary of the rotating tube into two parts, a leading edge and a trailing edge. The tube gains sites at the leading edge and loses sites at the trailing edge. Note that the tube center moves monotonically to the right during a clockwise rotation. Therefore, the trailing edge is that part of the tube's boundary from $(x_t - 1, \infty)$ to p, and the leading edge is that part from p to $(x_t + 1, \infty)$. In traversing the tube's boundary from $(x_t - 1, \infty)$ to $(x_t + 1, \infty)$, one moves monotonically from left to right. Therefore, the trailing edge is to the left of p and the leading edge is to the right. Similarly, during a counterclockwise rotation, the trailing edge is to the right of p and the leading edge is to the left. ∎

Lemma A.2 *A test tube contains a site if and only if it contains a maximal site.*

Proof: Suppose a tube T contains a site, and begin raising T. In the process, previously contained sites may drop out the bottom, but no new sites will enter. Continue raising

T until it contains only sites on its boundary, as shown in Figure A.2. Let p be the

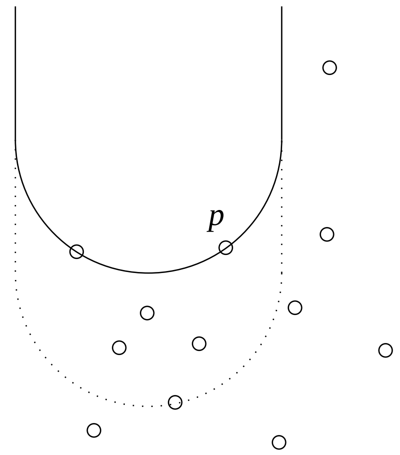


Figure A.2: Raising a test tube until it contains sites only on its boundary. The leftmost and rightmost such sites are necessarily maximal.

rightmost such site, and rotate T slightly clockwise about p. The rotated, raised test tube is now supporting, and p is maximal by definition.

The other direction is trivial since maximal sites are sites. ∎

Lemma A.3 *If a site p is incident to a test tube that contains a site to the left of p and a site to the right of p, then p is not maximal.*

Proof: The site p could not have a supporting tube. This is because any tube T incident to p can be rotated about p into any other incident tube. By the Rotation Lemma, any site in T to the left of p cannot be eliminated from T by rotating it counterclockwise. Similarly, any point to the right of p cannot be eliminated by rotating T clockwise. Therefore, if T contains sites on both sides of p, then any other incident tube contains one or the other, and therefore cannot be supporting. ∎

Lemma A.4 (The Neighbour Lemma) *Two sites p and q are neighbouring maxima if and only if the test tube through p and q contains only nonmaxima, and these only on the portion of its boundary between p and q.*

Proof: Let $p = (x_p, y_p)$ and $q = (x_q, y_q)$ be two sites and let T_{pq} be the tube through them. Assume without loss of generality that $x_p \leq x_q$. First assume that p and q are neighbouring maxima. Then each has a supporting tube T_p and T_q respectively, as shown in Figure A.3. Rotate T_p (clockwise) into T_{pq} about p. In the process, no sites can enter

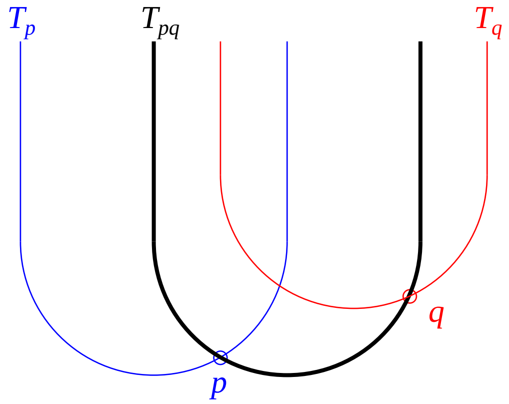


Figure A.3: Two neighbouring maxima p and q with supporting tubes T_p and T_q, and a tube T_{pq} through p and q.

T_p from the left of p by the Rotation Lemma. Therefore T_{pq} contains no sites left of p. By a symmetric argument with T_q, test tube T_{pq} contains no sites to the right of q. Now let r be the first site that entered T_p during its clockwise rotation into T_{pq}. If more than one site entered T_p at the same time, let r be the rightmost such site. Then $r = q$ and the rotated T_p is equal to T_{pq}. To see this, rotate T_p slightly clockwise about r at the moment r enters it. The rotated T_p is now supporting and r must be maximal. Since T_p contains no sites left of p nor right of q, and since there are no maxima between p and q, it must be that $q = r$. It follows that all sites that enter the rotating T_p do so "at the last instance", so that T_{pq} contains sites only on its boundary, and these must be nonmaxima between p and q.

Conversely, assume that T_{pq} contains only nonmaxima on the boundary between p and q. Rotate T_{pq} slightly counterclockwise about p, causing all sites other than p to drop out. The rotated tube T_{pq} therefore supports p, and certifies that p is maximal.

Similarly, tube T_{pq} rotated clockwise about q certifies that q is maximal. Now let r be a site between p and q. If r is in T_{pq}, then it is not maximal by assumption. Otherwise, it must lie below T_{pq}. Lower T_{pq} onto r so that Lemma A.3 applies to show that r is not maximal. Therefore there are no maxima between p and q, which makes p and q neighbouring maxima. ▮

Lemma A.5 *Let p be a maximal and let T_p be any incident test tube. If T_p does not contain the right neighbour of p, then T_p does not contain* any *sites to the right of p. Similarly, if T_p does not contain the left neighbour of p, then T_p does not contain any sites to the left of p.*

Proof: Let q be the right neighbour of p, and let T_{pq} be the tube through p and q. By the Neighbour Lemma, T_{pq} contains sites only on the boundary between p and q. Now rotate T_{pq} into T_p about p. If the rotation is clockwise, then T_p contains q by the Rotation Lemma. So if T_p does not contain q, the rotation must be counterclockwise, and any intermediate sites on the boundary of T_{pq} are lost. Therefore T_p cannot contain any sites to the right of p. ▮

Throughout the rest of this appendix, let L and R be two sets of sites in the plane such that L is strictly to the left of R. For convenience of expression, refer to ∂L and ∂R as *local maxima* and $\partial(L \cup R)$ as *global maxima.*

Definition A.6 The *bridge between ∂L and ∂R* is the pair (b_l, b_r), where b_l is the rightmost global maximal in ∂L, and b_r is the leftmost global maximal in ∂R. The bridge partitions ∂L and ∂R into four pieces (L_l, L_r, R_l, and R_r), where

$$\begin{aligned}
L_l &= \{p : p \in \partial L \text{ and } x_p \leq x_{b_l}\}, \\
L_r &= \{p : p \in \partial L \text{ and } x_{b_l} < x_p\}, \\
R_l &= \{p : p \in \partial R \text{ and } x_p < x_{b_r}\}, \text{ and} \\
R_r &= \{p : p \in \partial R \text{ and } x_{b_r} \leq x_p\}.
\end{aligned}$$

∎

Observation A.7 *All sites in L_l and R_r are global maxima. That is, $(L_l \cup R_r) \subseteq \partial(L \cup R)$.*

Proof: Let us begin by proving that $L_l \subseteq \partial(L \cup R)$. Clearly, $b_l \in L_l$ (and $b_r \in R_r$) are global maxima by definition. So assume $l' \in L_l \setminus \{b_l\}$, and let $T_{l'}$ be its local supporting tube. By definition, $T_{l'}$ does not contain any other sites from L; in particular, it does not contain b_l. Suppose, to reach a contradiction, that $T_{l'}$ contains a site $r' \in R$. Like all sites in R, site r' lies to the right of b_l. The maximal b_l must therefore lie below the tube $T_{l'}$. Lower $T_{l'}$ onto b_l and notice that $T_{l'}$ still contains l' and r'. Then b_l is not maximal by Lemma A.3, a contradiction. Therefore $T_{l'}$ does not contain any sites from $L \cup R$ (other than l'), so that $T_{l'}$ is globally supporting and l' is maximal. By symmetry, all local maxima right of b_r are also global maxima. In conclusion, $(L_l \cup R_r) \subseteq \partial(L \cup R)$. ∎

Observation A.8 *No sites in L_r or R_l are global maxima.*

Proof: A global maximal $p \in L_r$ would contradict the definition of b_l as the *rightmost* global maximal in ∂L. Similarly, a global maximal $p \in R_l$ would contradict the definition of b_r as the *leftmost* global maximal in ∂R. ∎

The observations above show how local maxima relate to global maxima. Since every global maximal is also a local maximal, they (Observation A.7 and A.8) lead immediately to the following lemma.

Lemma A.9 (The Bridge Lemma) *If (b_l, b_r) is the bridge between ∂L and ∂R, then the set of global maxima is equal to the union of the local maxima left of b_l and the local maxima right of b_r inclusive, that is, $\partial(L \cup R) = (L_l \cup R_r)$.*

A.3 The Data Structure

The *test tube data structure* $(B(S), D(S))$ for a set S of sites consists of two parts: a (dynamic) balanced tree $B(S)$, and a (more involved) static binary tree $D(S)$ that we will refer to as a *deletion tree*. Let us examine the balanced tree $B(S)$ part of the data structure first. This balanced tree $B(S)$ stores only the maxima of the point set S. A standard balanced tree will suffice. For example, a red-black tree (*cf.* [Sed83], and see also [CLR90] pages 263–280), takes $O(h)$ space, where $h \leq |S|$ is the number of maxima of the sites in S, and supports binary search, insertion, and deletion, all in $O(\log h)$ time. The preprocessing algorithm for the data structure first constructs the list of maxima, then builds the balanced tree $B(S)$ by inserting one maximal at a time. The preprocessing algorithm therefore spends $O(h \log h)$ time dealing with the balanced tree. The algorithm constructs the list of h maxima in $O(S \log S)$ time using the deletion tree; this will be described shortly.

At the highest level, the deletion tree $D(S)$ is a complete static binary tree of height $\lceil \log |S| \rceil$. It has $2^{\lceil log |S| \rceil} = \Theta(S)$ leaves, $|S|$ of which store the sites S, sorted by x-coordinate. Let $site[v]$ denote the site (or NIL) stored at leaf v. The variable $root[D]$ stores a pointer to the root of the tree, and the variable $lchild[v]$ (respectively $rchild[v]$) stores a pointer to the left (respectively right) child of the internal node v. An internal node of $D(S)$ represents the (local) maxima of the sites stored in the leaves of its subtree. For space efficiency, each internal node stores a list of only those maxima that do not appear in any higher nodes; $Q[v]$ stores the list for node v. More specifically, if ∂L and ∂R are the maxima of the left and right subtrees of a node v, and (b_l, b_r) is the bridge between them, then $Q[lchild[v]]$ stores L_r and $Q[rchild[v]]$ stores R_l (recall Definition A.6).

In addition, every node stores the least $inf[v]$ and greatest $sup[v]$ x-coordinates of the leaves of its subtree. At this stage, note that the sites in the leaves of the subtrees rooted at the children of an interior node v can be separated by the vertical line $x = (sup[lchild[v]] + inf[rchild[v]])/2$, and that this value x is computable in constant time. This "separator" allows binary search to find a site in $O(\log S)$ time.

In summary, $D(S)$ has $O(S)$ leaves and just as many (less one) internal nodes. The maxima stored at the internal nodes partition the sites S, and the red-black tree at the root is also of linear size, so the entire data structure uses only $O(S)$ space.

Each node in the deletion tree stores its maxima as a linked list with elements ordered from left to right. Standard linked list techniques manipulate these lists. In particular, they split and slice lists in constant time, given a pointer to the split point. More concretely, the algorithms below use the following constant-time primitives.

LIST(p) creates and returns the list containing just the pointer p.

SPLIT(L, p) returns the pair (L_l, L_r), where p is a pointer to an element of the list L, L_l is the prefix of the list S up to and including element p, and L_r is the suffix of the list L past p, but not including p.

SPLICE(L, R) concatenates list R to the end of list L and returns a pointer to the result.

Each linked list also maintains pointers to its leftmost $first[h]$ and rightmost $last[h]$ elements. In the following algorithms, no site will ever be in more than one linked list at a time. We can therefore use the leaves of the deletion tree as list elements; each leaf v has a pointer to its successor (right neighbour) $succ[v]$ and to its predecessor (left neighbour) $pred[v]$ in the list containing site v. In this way, even an isolated pointer to a list element will always point to the same site, no matter how often the element has participated in splits and splices.

Finally, to allow easy reconstruction of the maxima represented at each node, each internal node stores a pointer to its bridge. More specifically, if ∂L and ∂R are the maxima of the left and right subtrees of a node v, and (b_l, b_r) is the bridge between them, then $bridge[v] = b_l$ (again, recall Definition A.6). It is easy to see that if ∂S is the list of maxima at an internal node v, then the maxima list at $lchild[v]$ is SPLICE(L_l, L_r), where (L_l, R_r) = SPLIT(∂S, $bridge[v]$) and $L_r = Q[lchild[v]]$. Similarly, the maxima list at $rchild[v]$ is SPLICE(R_l, R_r), where $R_l = Q[rchild[v]]$. These operations are encapsulated by Algorithm CHILDREN in Table A.1, which returns two lists of maxima, one for each subtree rooted at the children of an internal node v in $D(S)$, given a list of the maxima of the subtree rooted at v. It is easy to verify that Algorithm CHILDREN returns these lists in constant time.

Table A.1: **Algorithm:** CHILDREN(v, $\partial(L \cup R)$)

Input: The global maxima $\partial(L \cup R)$ of the sites stored in the leaves of the subtree rooted at v.
Output: The lists ∂L and ∂R of the local maxima of the sites stored in the leaves of the subtrees rooted at $lchild[v]$ and $rchild[v]$ respectively.

$\partial L \leftarrow \partial R \leftarrow$ NIL
$l \leftarrow bridge[v]$
if $l \neq$ NIL
  then $(L_l, R_r) \leftarrow$ SPLIT($\partial(L \cup R)$, l)
    $\partial L \leftarrow$ SPLICE(L_l, $Q[lchild[v]]$)
    $\partial R \leftarrow$ SPLICE($Q[rchild[v]]$, R_r)
  else $\partial L =$ NIL
    $\partial R = \partial(L \cup R)$
return $(\partial L, \partial R)$

A.4 Building the Data Structure

The preprocessing algorithm constructs $D(S)$ by first building the tree skeleton, then filling in the leaves, and finally filling in the internal nodes. The first step is standard binary tree construction, and takes $O(S)$ time. The algorithm then fills in the leaves by sorting the sites in $O(S \log S)$ time and putting them in the leaves in $O(S)$ time. In the process, the algorithm also sets the fields *inf*$[v]$ and *sup*$[v]$ that point to the leftmost and rightmost extreme leaves in the subtree rooted at v. Clearly, $inf[v] = inf[lchild[v]]$ and $sup[v] = sup[rchild[v]]$. The algorithm therefore accomplishes both tasks with a postorder traversal of the tree.

The algorithm then fills in the internal nodes of $D(S)$. To construct the list of maxima at the root, the algorithm recursively constructs the maxima of its two children and merges them. The details for Algorithm FILL-INTERNAL are presented in Table A.2 below.

Table A.2: **Algorithm:** FILL-INTERNAL(v)

Input: A node v in the tree D.
Output: A doubly linked list of the maxima of the leaves of the subtree rooted at v.

if v is a leaf
  then return $LIST(v)$ (with $pred[v] = succ[v] =$ NIL)
  else $\partial L \leftarrow$ FILL-INTERNAL($lchild[v]$)
      $\partial R \leftarrow$ FILL-INTERNAL($rchild[v]$)
      $l \leftarrow first[\partial L]$
      $r \leftarrow last[\partial R]$
      return MERGE(v, ∂L, l, ∂R, r)

Lemma A.10 *Algorithm FILL-INTERNAL(v) (Table A.2) computes the maxima of the leaves of the subtree rooted at any node v of the deletion tree in $O(n \log n)$ time, where n is the number of sites stored in the leaves of the subtree rooted at v.*

Proof: The proof is by induction on the minimum path length from v to a leaf. The lemma is trivially true if v is a leaf. So assume that v is an internal node, and let L

and R denote the sites stored at the leaves of v's left and right subtrees. We now wish to compute the global maxima of L and R, where R is strictly to the right of L. Since all global maxima are local maxima, it suffices to look only at local maxima. By the inductive hypothesis, the lists ∂L and ∂R, computed at Steps 3 and 4, are the maxima of L and R, as required. Step 5 computes the leftmost local maximal l in ∂L, and Step 6 computes the rightmost local maximal r in ∂R. These values, l and r, are global maxima since l is supported by the tube centered at $(x_l - 1, y_l)$, and r is supported by the tube centered at $(x_r + 1, y_r)$. By Lemma A.12 (forthcoming), the call to MERGE(v, ∂L, l, ∂R, r) therefore returns the global maxima $\partial(L \cup R)$ in $O(h)$ time, where $h = |\partial(L \cup R)|$ is the number of global maxima. Clearly, h is never more than the number n of sites stored at the leaves of the subtree. All other steps take constant time. The time taken for a call to FILL-INTERNAL(*root*[D]) is therefore given by the recurrence $T(n) = n + 2T(n/2)$, which has solution $O(n \log n)$ time. ∎

The complete algorithm for building the deletion tree data structure therefore builds the skeleton of the tree in linear time, sorts the sites from left to right in $O(S \log S)$ time, and finally executes

$$Q[root[D]] \leftarrow \text{FILL-INTERVAL}(root[D])$$

to fill in the distributed lists of maxima in the internal nodes, also in $O(S \log S)$ time. This establishes the following theorem.

Theorem A.11 *The test tube data structure $(B(S), D(S))$ of sites S can be built in $O(S \log S)$ time and $O(S)$ space.*

We must now address the problem of efficiently "merging" two lists ∂L and ∂R to yield a list of global maxima $\partial(L \cup R)$. By the Bridge Lemma, we need only find the bridge points, split the local lists at these points, and splice the global maxima together.

Table A.3 provides the details. Of course, if either ∂L or ∂R is an empty list, there is no bridge (b_l, b_r). Let us indicate this condition by setting $bridge[v]$ to NIL. Algorithm MERGE is also needed later in this appendix to handle deletions from the deletion tree. To facilitate this, Algorithm MERGE maintains the maxima tree by storing lists of local maxima where appropriate. Again, see Table A.3 for details.

Table A.3: **Algorithm:** MERGE(v, ∂L, l, ∂R, r)

Input: The left maxima ∂L and right maxima ∂R of internal node v, and global maxima $l \in \partial L$ and $r \in \partial R$.
Output: The list of global maxima $\partial(L \cup R)$.
Side Effect: The algorithm maintains the deletion tree D by storing the list of sites in $\partial L \setminus \partial(L \cup R)$ at the left child of v, and the list of sites in $\partial R \setminus \partial(L \cup R)$ at the right child of v.

1 **if** either ∂L or ∂R is empty,
2   **then** $bridge[v] \leftarrow$ NIL
3         $(L_l, L_r) \leftarrow (\partial L, \text{NIL})$
4         $(R_l, R_r) \leftarrow (\text{NIL}, \partial R)$
5   **else** $(b_l, b_r) \leftarrow$ BRIDGE(∂L, l, ∂R, r)
6         $bridge[v] \leftarrow b_l$
7         $(L_l, L_r) \leftarrow$ SPLIT(∂L, b_l)
8         $(R_l, R_r) \leftarrow$ SPLIT(∂R, $pred[b_r]$)
9 $Q[lchild[v]] \leftarrow L_r$
10 $Q[rchild[v]] \leftarrow R_l$
11 **return** SPLICE(L_l, R_r)

Except for the call to function BRIDGE (Step 5), all other operations in Table A.3 take constant time. Algorithm BRIDGE finds the bridge simply by "walking" along list ∂L until it reaches the rightmost maximal, and along ∂R until it reaches the leftmost maximal. The correctness of Algorithm BRIDGE in Table A.4 follows from the Bridge Lemma.

How can Algorithm BRIDGE decide if l' and r' (Steps 2 and 3) are global maxima,

Table A.4: **Algorithm:** BRIDGE(∂L, l, ∂R, r)

Input: Local maxima ∂L and ∂R,
global maxima $l \in \partial L$ and $r \in \partial R$.
Output: The bridge (b_l, b_r).

while $succ[l]$ or $pred[r]$ is a global maximal,
    do $l' \leftarrow succ[l]$ /* the right neighbour of l in ∂L, if it exists. */
        $r' \leftarrow pred[r]$ /* the left neighbour of r in ∂R, if it exists. */
        if l' is a global maximal
            then $l \leftarrow l'$
                else if r' is a global maximal
                    then $r \leftarrow r'$
return (l, r).

and how quickly? The answer is that it can determine this in constant time, as the following case analysis demonstrates. Let T_{lr} be the test tube through l and r. If both l' and r' are NIL, both are clearly not maximal and we are done. So assume that either l' or r' is not NIL. If l' lies strictly inside T_{lr}, but r' does not (perhaps because it is NIL), then l' is maximal. To see this, first note that T_{lr} does not contain any sites from R left of r by Lemma A.3 and Lemma A.5. Rotate T_{lr} into $T_{ll'}$ about l, in the process dropping r below $T_{ll'}$. Now lower $T_{ll'}$ onto r so that $T_{ll'}$ still contains l and l' (to the left of r). Since r is a global maximal, Lemma A.3 implies that the lowered tube does not contain any sites to the right of r. Therefore the unlowered $T_{ll'}$ does not contain any sites from R, and l' is a global maximal by the Neighbour Lemma. By a symmetric argument, r' is maximal if r' lies strictly inside T_{lr}, but l' does not.

The final case is that both l' and r' lie properly within T_{lr}. Let $T_{ll'}$ be the tube through l and l'. If $T_{ll'}$ is strictly left of the separator, then l' is maximal. To see this, rotate $T_{ll'}$ slightly clockwise about l', dropping l in the process. It now does not contain any sites from L (other than l') by the Neighbour Lemma. It does not contain any sites from R by the properties of the separator. It is therefore supporting with respect to $L \cup R$ so that l' is maximal. Similarly, if the tube $T_{r'r}$ through r and r' is completely to the right of the separator, then r' is maximal.

Finally, suppose that both $T_{ll'}$ and $T_{r'r}$ intersect the separator, as shown in Figure A.4. Since l' and r' both lie inside T_{lr}, then $T_{ll'}$ also lies to the left of $T_{r'r}$. To see this, note

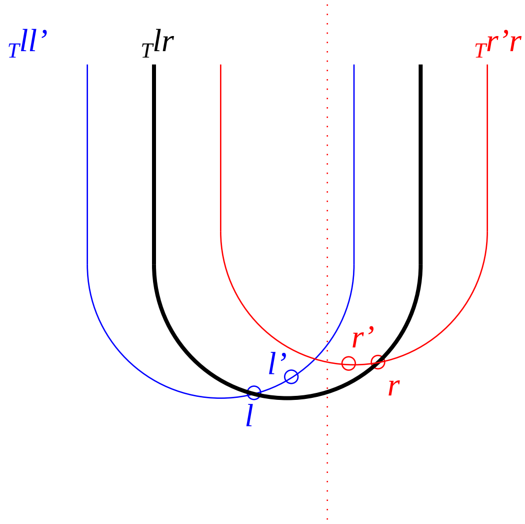


Figure A.4: Both $T_{ll'}$ and $T_{r'r}$ intersect the separator. In the case shown here, both l' and r' lie inside the tube T_{lr} through l and r.

that $T_{ll'}$ can be rotated clockwise[3] into T_{lr}, and T_{lr} can be rotated clockwise in $T_{r'r}$. The separator therefore lies between the left wall of $T_{r'r}$ on the left and the right wall of $T_{ll'}$ on the right. If $T_{ll'}$ falls above $T_{r'r}$ at the separator, as shown in Figure A.4, then l' is maximal. To see this, note that the only sites from L in $T_{ll'}$ lie on the boundary of $T_{ll'}$ between l and l' by the Neighbour Lemma. Rotating $T_{ll'}$ slightly clockwise about l' therefore drops all such sites from $T_{ll'}$. No sites from R can be in $T_{ll'}$ since the portion of $T_{ll'}$ that lies to the right of the separator lies in the interior of $T_{r'r}$, which is empty of sites from R by the Neighbour Lemma. By a symmetric argument, if $T_{r'r}$ falls above $T_{ll'}$

[3]Recall that a clockwise rotation moves a tube monotonically to the right.

at the separator, then r' is maximal, and both l' and r' are maximal if the tubes coincide at the separator.

Since Algorithm BRIDGE (Table A.4) can determine in constant time if l' or r' is a maximal, the algorithm spends a constant amount of time at each step. Other than the final nonmaximal l' and r', the algorithm only examines global maxima. If it does not return the bridge at a given step, it eliminates a global maximal from further consideration. The algorithm therefore finds the bridge (a, b) in $O(h)$ time. Here h is the number of sites in ∂L between l and a, plus the number of sites in ∂R between b and r. This completes the description of Algorithm MERGE, and we have established the following lemma.

Lemma A.12 *Let L and R be two horizontally separated sets of sites. Given are the left-to-right ordered linked lists ∂L and ∂R, and two sites $l \in \partial L$ and $r \in \partial R$ guaranteed to be maximal in $L \cup R$. Algorithm MERGE (Table A.3) computes the list $\partial(L \cup R)$ of global maxima in $O(h)$ time, where h is the number of global maxima $\partial(L \cup R)$ between l and r.*

A.5 Deletions

How does the deletion tree $D(S)$ support deletions in $O(\log S)$ amortized time? To answer this question, we will first examine how a set of maxima ∂S changes when a site p is deleted from S. Lemma A.14 (below) shows that, if a site p is deleted from a set of sites and p is not maximal, then the list of maxima does not change. Even if p is maximal, Lemma A.13 (below) shows that no maxima other than p are "lost" by deleting p from S. In this case, however, new maxima may be created. Where could these maxima lie? The answer is that the new maxima must lie between the neighbours of the deleted site (Lemma A.15 below).

Lemma A.13 *Let p be any site in S. All maxima in S (other than p) are also maximal in $S \setminus \{p\}$. That is, $\partial S \setminus \{p\} \subseteq \partial(S \setminus \{p\})$.*

Proof: Let q be a maximal (other than p) in S. Its supporting tube T_q is empty of any other sites in S, and therefore of any sites in $S \setminus \{p\}$. Tube T_q is therefore supporting with respect to $S \setminus \{p\}$, which certifies that q is maximal in $S \setminus \{p\}$. ∎

Lemma A.14 *Let p be a nonmaximal site in S. The maxima in $S \setminus \{p\}$ are exactly the maxima in S. That is, $\partial(S \setminus \{p\}) = \partial S$.*

Proof: Lemma A.13 implies that $\partial S \subseteq \partial(S \setminus \{p\})$.

To prove that $\partial(S \setminus \{p\}) \subseteq \partial S$, let q be a site in $\partial(S \setminus \{p\}$. Then q has an incident tube T_q that is supporting with respect to $S \setminus \{p\}$. Assume for the sake of contradiction that q is not maximal in S. Then T_q must contain a site from S, and therefore a maximal $r \in S$ by Lemma A.2. Since neither p nor q are maximal, r is a site other than q in $S \setminus \{p\}$. Therefore T_q is not supporting with respect to $S \setminus \{p\}$, a contradiction. ∎

Lemma A.15 *Let p be a maximal site in S. If q is a maximal site in $S \setminus \{p\}$ but not in S, then q must lie between the left neighbour of p and the right neighbour of p.*

Proof: Let T_q be a supporting tube for q with respect to $S \setminus \{p\}$. Note that T_q must contain p, otherwise q would be maximal in S. Now suppose that q lies to the left of the left neighbour l of p. Then l is between q and p, and therefore below T_q, as shown in Figure A.5. Lower tube T_q onto l and notice that it still contains both q and p. Therefore l is not maximal by Lemma A.3, contradicting the definition of neighbour. Similarly, if q is right of r, then r could not be maximal, again a contradiction. ∎

In terms of our deletion tree data structure $D(S)$, maxima move to higher levels in the tree in response to site deletions until they are themselves deleted. Since the data

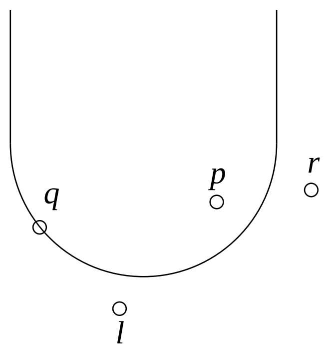


Figure A.5: Site q is maximal in $S \setminus \{p\}$ but not in S. Assume that q lies to the left of the left neighbour l of p.

structure has $O(\log S)$ levels, there are a total of $O(S \log S)$ site movements between levels.

Algorithm DELETE-SITE (Table A.5) implements deletions recursively. From an interior node, it deletes the site from the appropriate child's subtree—it can decide which subtree in constant time by comparison with the stored separator—and merges the new list with the unchanged list of the other subtree. As we already illustrated, a merge takes constant time once we have a pointer to the bridge. We now show that DELETE-SITE finds this bridge in constant amortized time.

Let $\partial(L \setminus \{p\})$ be the maxima in the left subtree after having recursively deleted p, and let ∂R be the maxima in the right subtree. If p was not maximal at this internal node, then the current bridge is still valid by Lemma A.14. So assume that p was maximal. If p is not the bridge site l in ∂L or its neighbour r in ∂R, then the bridge is still valid by the Neighbour Lemma.

Now suppose that the deleted site p is l. See Figure A.6. Let l' be the right neighbour of l and let l'' be the left neighbour of l, both in ∂L. Similarly, let r' be the left neighbour of r and let r'' be the right neighbour of r, both in ∂R. Any new maxima will appear between l'' and r by Lemma A.15. By Lemma A.13, all maxima in $L \cup R$ are maxima in

Table A.5: **Algorithm:** DELETE-SITE(p, $\partial(L \cup R)$, v)

Input: A node v,
a list $\partial(L \cup R)$ of the maxima of the sites stored at the leaves of the subtree rooted at v, and
a site p to be deleted.

Output: A list of the maxima of the sites stored at the leaves of the subtree rooted at v after p is deleted.

Side Effect: The site p is deleted from the subtree rooted at v.

if v is a leaf
    then $site[v] \leftarrow$ NIL
          return NIL
    else $l \leftarrow root[v]$
          $l' \leftarrow pred[l]$
          $r \leftarrow succ[l]$
          $r' \leftarrow succ[r]$
          $(\partial L, \partial R) \leftarrow$ CHILDREN(v, $\partial(L \cup R)$)
          if $x[p] \leq sup[lchild[v]]$
            then $\partial L \leftarrow$ DELETE-SITE(p, ∂L, $lchild[v]$)
                    return MERGE(v, ∂L, l'', ∂R, r)
            else $\partial R \leftarrow$ DELETE-SITE(p, ∂R, $rchild[v]$)
                    return MERGE(v, ∂L, l, ∂R, r'')

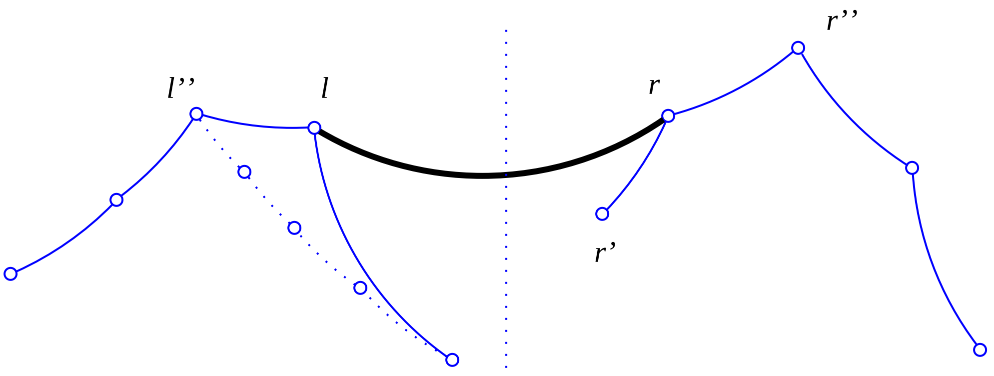


Figure A.6: Deleting the left bridge site, l.

$(L \cup R) \setminus \{l\}$. In particular, l'' is maximal in $(L \cup R) \setminus \{l\}$ so the new bridge point in L cannot lie to the left of l''. Similarly, the right neighbour (in R) of the new bridge point cannot lie to the right of r.

We can therefore use Algorithm MERGE(v, $\partial(L \setminus \{l\})$, l'', ∂R, r) to compute $\partial((L \setminus \{l\}) \cup R)$ in $O(h)$ time. Here h is the number of new maxima, ones that have risen from the level below. We can "charge" this search to site movements. Similarly, if the deleted point is r, we can use Algorithm MERGE(v, ∂L, l'', $\partial(R \setminus \{r\})$, r) to compute $\partial(L \cup (R \setminus \{r\}))$, also in $O(h)$ time.

Since there are $O(\log S)$ levels to the tree, each deletion takes $O(\log S)$ time, not including site movements. We can therefore execute $O(S)$ deletions in $O(S \log S)$ time, plus $O(S \log S)$ for site movements, for a total time of $O(S \log S)$. We have established the following lemma.

Lemma A.16 *Sites $\{p_1, p_2, \ldots, p_n\} \subseteq S$ can be deleted from the deletion tree $D(S)$ in $O(S \log S)$ time by executing Algorithm DELETE-SITE(p_i, $Q[\mathrm{root}[D(S)]]$, $\mathrm{root}[D(S)]$) (Table A.5) for every i from 1 to n.*

We now know how to delete a site from the deletion tree part of the data structure. However, a deletion algorithm must also maintain the balanced tree $B(S)$ part of the data structure. If the deleted site is not maximal, then the maxima do not change by Lemma A.14, so the deletion algorithm does not need to do anything with $B(S)$. If the site is maximal, and therefore in $B(S)$, then this site is the only one that the algorithm has to delete from $B(S)$, by Lemma A.13. Since each site can be deleted at most once, these balanced tree deletions take a total of $O(S \log S)$ time.

On the other hand, some previously nonmaximal sites may become maximal due to site deletion. These new maxima are easily isolated. If p_l and p_r are the left and right neighbours of p in $Q[root[D]]$ before p is deleted, then the new maxima are the sites between p_l and p_r in $Q[root[D]]$ after p is deleted, by Lemma A.15. The site deletion algorithm executes an insertion whenever a new maximal rises to the root of D. Since each site can rise to the root at most once, this takes a total of $O(S \log S)$ time. The following theorem summarizes the preceding discussion.

Theorem A.17 *$O(S)$ sites can be deleted from the test tube data structure $(B(S), D(S))$ of sites S in $O(S \log S)$ time.*

Index