Leveled Drawings of Graphs

A thesis submitted for the degree of

Doctor of Philosophy

by

Martin Harrigan

Dept. of Computer Science and Information Systems

University of Limerick

Supervisor: Dr. Patrick Healy

Submitted to the University of Limerick, June 2008.

Declaration

I hereby declare that this thesis is entirely my own work, and that it has not been submitted as an exercise for a degree at any other University.

Acknowledgments

Even though this thesis can yield at most one doctorate to at most one person, it has necessitated the advice, help and support of many. It seems only appropriate to thank them at the very beginning.

My supervisor, Dr. Patrick Healy, has contributed enormously to this work in terms of intellectual input and support. He has given generously of his time and insights, and not a word of mine has appeared in print or in this dissertation which he has not read and reflected and commented on. He has been able to individuate what I am best at and given me the guidance and time to make sense of everything else.

I wish to thank all my colleagues at the University of Limerick. For their expressions of best wishes and inquiries into the progress I was or was not making, or for simply listening to my rants about the latter, I thank Karol Lynch, Nikola Nikolov, Aimal Rextin and Radoslav Andreev.

I am grateful to Dr. Ulrik Brandes for hosting my visit to the Algorithmics Group at the University of Konstanz. It was an edifying and stimulating experience to see his group at work. I am also especially grateful to Frau Dreher and Frau Hustert for providing the most cordial and comfortable accommodation during my stay.

I owe my gratitude to the Irish Research Council for Science, Engineering and Technology for providing funding throughout my studies under the Embark Initiative. Their travel grants allowed me to participate in many workshops and conferences abroad. I am also grateful to the Department of Computer Science and Information Systems for providing a studious and pleasant work environment.

Finally, I would like to thank my family for offering unconditional support at every turn. They provided me with the means to learn and understand and without them, I could not have followed this path. And last but not least Deirdre, not just for coming along at the right time, but for the very special person she is.

Abstract

A leveled graph is a graph $G = (V, E)$, with vertex set $V(G) = V_1 \cup V_2 \cup \ldots \cup V_k$, $V_i \cap V_j = \emptyset$, $i \neq j$, and edge set $E(G) = E_1 \cup E_2 \cup \ldots \cup E_{k-1}$ such that $E_i \subseteq V_i \times V_{i+1}$. A leveled drawing is a drawing of a leveled graph G in which the vertices of each V_i are placed on a horizontal line $y = ci$ and the edges are drawn as straight-line segments between the vertices.

The multi-level crossing minimization problem consists of finding a vertex ordering of each V_i so that the number of edge crossings in the leveled drawing is minimal. For the special case of level planar (crossing-free) drawings, we extend the work of Healy and Kuusik (2004). We show how to determine the orderings in quadratic running-time and handle a family of embedding constraints. We relate the notion of the vertex-exchange graph to signed graphs thereby applying an existing body of algorithms to the multi-level crossing minimization problem.

An alternative to crossing minimization is to expose a maximum level planar subgraph. We look at the special case of trees. We review methods of solving the crossing minimization and maximum level planar subgraph problems for two levels and show that they can also be solved easily when the order of the vertices on one of the levels is fixed through the use of crossing and planarizing penalty digraphs. We show that the maximum planarization problem for two levels is equivalent to the Hamiltonian cycle completion problem, for which there exist efficient algorithms for some special classes of graphs. Finally, we prove the NP-Hardness of the crossing minimization problem (for trees) when there are only two levels with the order of the vertices on one of the levels subject to some partial order and also for an arbitrary number of levels.

Finally, we present a method of drawing a directed graph that emphasizes a significant spanning tree. The spanning tree is a tree DAG with multiple sources and so it is more appropriate to give each of these vertices root status than to nominate any single vertex and hang the subtrees from it. We ensure the level planarity of the spanning tree by inserting dummy vertices and restricting the possible leveled drawings so that no two significant edges cross. We also show that finding a minimum number of dummy vertices to ensure level planarity even when the tree DAG has exactly one vertex of in-degree greater than one is NP-Hard.

Contents

1 Introduction 1

1.1 Graph Drawings 2

1.1.1 Leveled Drawings and the Sugiyama Framework 5

1.1.2 Other Methods of Drawing Leveled Graphs 7

1.2 Contributions and Organization 8

2 Preliminaries 11

2.1 Graph Theory 11

2.1.1 Undirected Graphs 11

2.1.2 Directed Graphs 13

2.1.3 Matrix Representations 15

2.2 Orders 16

2.3 Levelings and Level Planarity 17

2.4 Signed Graphs and the Vertex-Exchange Graph 20

2.5 The Parameterized Line Graph 21

2.6 Force-Directed and Energy Minimization Methods 28

3 Level Planarity 31

3.1 Previous Work 31

3.2 Testing and Embedding 33

3.2.1 The 3-Cycle Problem 37

3.2.2 The 3-Cycle Solution 39

3.2.3 Proof of Correctness 46

3.3 Extensions 53

3.3.1 Cyclic Level Planarity Testing and Embedding 54
3.3.2 Embedding Constraints 54
3.3.3 Multi-Level Crossing Minimization 58

4 Crossing Minimization and Maximum Planarization 63

4.1 Previous Work 65
4.2 Crossing Minimization for Leveled Trees 67
4.2.1 2-Leveled Trees 67
4.2.2 2-Leveled Trees with One Level Fixed 68
4.2.3 k-Leveled Trees 70
4.3 Maximum Level Planar Subgraph for Leveled Trees 74
4.3.1 2-Leveled Trees 75
4.3.2 2-Leveled Trees with One Level Fixed 76

5 Ensuring Level Planarity 81

5.1 Choosing a Significant Spanning Tree DAG 82
5.2 Ensuring Level Planarity of a Leveled Tree 84
5.2.1 The Base Case 84
5.2.2 The Recursive Case 91
5.2.3 Proof of Correctness 94
5.2.4 Implementation 95
5.2.5 A Worked Example 96
5.2.6 Violating the Leveling 99
5.3 Crossing Minimization 104
5.4 Bringing It All Together 105

6 Conclusions and Open Problems 108

List of Figures

1.1 A drawing of a graph. 2

1.2 A leveled drawing of a QCA half-adder. 4

2.1 The transitive closure and transitive reduction of a directed graph. 14

2.2 A k-leveled drawing of a graph G with leveling ϕ. 18

2.3 A leveled graph, vertex-exchange graph and labeled vertex-exchange graph. 22

3.1 First contradiction in the proof of Theorem 3.2.1. 35

3.2 Second contradiction in the proof of Theorem 3.2.1. 37

3.3 The 3-cycle problem. 38

3.4 `switch`($\langle u, v\rangle$) when u and v have no vertices in between them. 41

3.5 `switch`($\langle u, v\rangle$) when u and v have vertices in between them. . . 41

3.6 A corrected labeled vertex-exchange graph. 42

3.7 Grouping the vertices of a labeled vertex-exchange graph. 44

3.8 Traversing a labeled vertex-exchange graph (Case 1). 46

3.9 Traversing a labeled vertex-exchange graph (Case 2). 46

3.10 Traversing a labeled vertex-exchange graph (Case 3). 47

3.11 Traversing a labeled vertex-exchange graph (Case 1 again). . . . 47

3.12 Classes of edges in the proof of Theorem 3.2.2. 49

3.13 Fewer classes of edges in the proof of Theorem 3.2.2. 49

3.14 Classes of edges in the proof of Theorem 3.2.2. 51

3.15 A labeled vertex-exchange graph in the proof of Theorem 3.2.2. 52

3.16 Classes of edges in the proof of Theorem 3.2.2. 52

3.17 A simpler labeled vertex-exchange graph in the proof of Theorem 3.2.2. 53
3.18 Exchanging u and v in $\mathcal{L}$ in the proof of Theorem 3.2.2. 53
3.19 A final labeled vertex-exchange graph in the proof of Theorem 3.2.2. 54
3.20 A cyclic k-leveled drawing of a graph G with cyclic leveling ϕ. . 55
3.21 A constraint tree. 56
3.22 Expanding a leveled graph with a constraint tree. 57
3.23 Expanding a leveled graph with a constraint tree. 58
3.24 A k-leveled drawing of a leveled graph with the minimum number of edge crossings. 61

4.1 A 2-leveled drawing of a tree with the minimum number of edge crossings. 68
4.2 A 3-leveled drawing of a tree with more edge crossings than necessary. 68
4.3 A construction in the proof of Corollary 4.2.3. 69
4.4 An example in the proof of Corollary 4.2.3. 70
4.5 A construction in the proof of Theorem 4.2.4. 71
4.6 An augmented tree in the proof of Theorem 4.2.4. 73
4.7 A 2-leveled drawing of a tree. 79
4.8 A planarizing penalty digraph. 80

5.1 Two spanning tree DAGs. 83
5.2 The base case in the algorithm `makeTreeDAGLevelPlanar`. . . . 85
5.3 A contradiction in the proof of Lemma 5.2.1. 86
5.4 Reducing a task to a group of directed trees in the proof of Theorem 5.2.5. 90
5.5 Reducing a set of tasks to a tree DAG in the proof of Theorem 5.2.5. 90
5.6 Smothering in the algorithm `makeTreeDAGLevelPlanar`. 92
5.7 The four MLNP tree patterns. 94

5.8 The four MLNP tree patterns with dummy vertices inserted. . . 95
5.9 Storing $M(T)$ in a sparse form. 96
5.10 A worked example for algorithm `makeTreeDAGLevelPlanar`. . . 97
5.11 A worked example for algorithm `makeTreeDAGLevelPlanar`. . . 99
5.12 A set of fundamental cycles. 100
5.13 Partitioning the edge set into edge subsets. 103
5.14 The dependency graph for Python packages. 107

List of Tables

4.1 Summary of results for the k-level crossing minimization problem for trees. 65

4.2 Summary of results for the k-level maximum level planar subgraph problem for trees. 65

List of Algorithms

1 `dfsTraversal` 40
2 `levelByLevelTraversal` 43
3 `makeTreeDAGLevelPlanar` 93

Chapter 1

Introduction

A *graph* is a mathematical structure typically used to model the relationships between objects. It consists of *nodes* or *vertices* linked together by (possibly *directed*, *weighted*) *edges*. A graph can be defined by a list of vertices and edges, by using a matrix structure or simply by a *drawing* of the graph.

A drawing of a graph is a visual representation of its vertices and edges. We describe one such representation (see Fig. 1.1). A vertex is depicted by a polygonal or circular shape with an optional label printed inside or close to the shape. An edge is depicted by a polyline or curve which connects the shapes and may also have an optional label. If an edge has an associated direction it is decorated with an arrow to indicate this direction. A drawing is a one-to-one mapping of each vertex to a shape and each edge to a polyline or curve in the plane. With respect to a particular drawing, we can refer to the elements of a graph or their representations interchangeably. For example, we say that two edges cross if their corresponding polylines or curves intersect in the drawing.

A 'good' drawing of a graph not only defines the graph but can effectively aid its comprehension by humans. The properties of a graph drawing that make it good are called *aesthetics*. These include minimizing the number of edges crossings, minimizing the area of the drawing, minimizing the sum of the edge lengths, keeping the edge lengths uniform, minimizing the number of edge bends and maximizing the size of the smallest angle formed by two edges

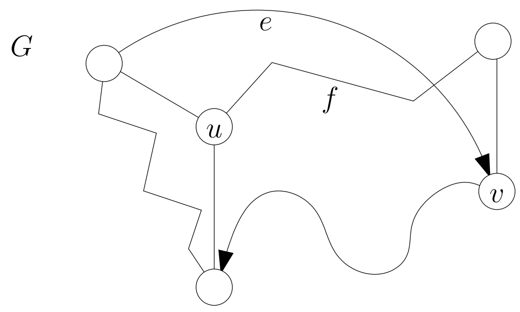


Figure 1.1: A drawing of a graph with five vertices and six edges. There are two labeled vertices, two labeled edges and two directed edges. The labeled edges, e and f, cross.

meeting at the same vertex. Aesthetics can conflict, *e.g.* increasing the number of edge bends can provide for shorter overall edge length. Graph drawings can be subject to *constraints* (Tamassia, 1998). Constraints allow certain aspects of a drawing to be fixed, *e.g.* drawing a subgraph with a certain shape. *Graph drawing algorithms* are methods to produce good drawings subject to these constraints (see Di Battista et al. (1999); Kaufmann and Wagner (2001) for surveys). These algorithms can be categorized according to the type of drawing they produce. This thesis is particularly interested in *leveled drawings* of graphs.

1.1 Graph Drawings

Graph drawing pervades many diverse fields. Kruja et al. (2002) document some early instances of pre-Eulerian graph drawing along with some modern examples from fields as diverse as topology, crystallography, chemistry and mathematical puzzles. We briefly describe some interesting application areas for graph drawing.

A recent development in the field of nano-technology draws from graph drawing techniques. *Quantum-dot cellular automata (QCA)* (Lent and Tougaw, 1997) provide a new way to represent binary information. They replace the current switch with a cell having a bi-stable charge configuration (representing a bit). These cells operate at the nanometer scale and rely on quantum mechanical effects to function. On the other hand, these same effects hinder the continual miniaturization of transistors. Grid arrangements of quantum-dot cells can perform computation. The simplest practical cell arrangement is the placement of quantum-dot cells in a series. If the polarization of any one of the cells can be controlled, then the rest of the cells immediately synchronize to its polarization. A complete set of logic gates can be constructed using the same principle.

The automatic design of QCA circuits (Lim and Niemier, 2004; Ravichandran et al., 2005; Chaudhary et al., 2007) involves a process almost identical to that of drawing *leveled graphs*. Constraints like uniform edge directionality, few edge crossings, short overall edge length, and a maximum width for the cells on each level all feature prominently. Figure 1.2 shows a left-to-right *leveled drawing* of a QCA half-adder with five levels. All the cells are initially unpolarized. If the three inputs on the left are fixed, then their polarizations propagate through the series of cells (the dots in each cell repel each other) and are combined at the junctions using logic gates to produce the outputs on the right. With QCA circuits, it appears to be very difficult to fabricate wire crossings in the near to mid-term. An interesting problem that arises in this application, which is somewhat related to leveled graphs, involves the elimination of all crossings with a minimal amount of logic gate/vertex duplication (Chaudhary et al., 2007).

Many more applications can be found in the literature, for example, database and software diagrams (Batini et al., 1986; Nummenmaa and Tuomi, 1990; Protsko et al., 1991; Doorley, 1995), schematic diagrams for electronic circuits (Aoudja et al., 1986), Hasse diagrams (Jourdan et al., 2004), ontologies (Katifori et al., 2007) and object-oriented class diagrams (Seemann, 1997; Eichelberger, 2005). In standard cell technology for VLSI (Lengauer, 1990; Sarrafzadeh and Wong, 1996), for example, modules are arranged in rows with wiring channels between each pair of rows. VLSI layouts containing fewer channel crossings are more easily realizable and consequently cheaper to produce. A *maximum level planar subgraph* represents a maximum subset of the

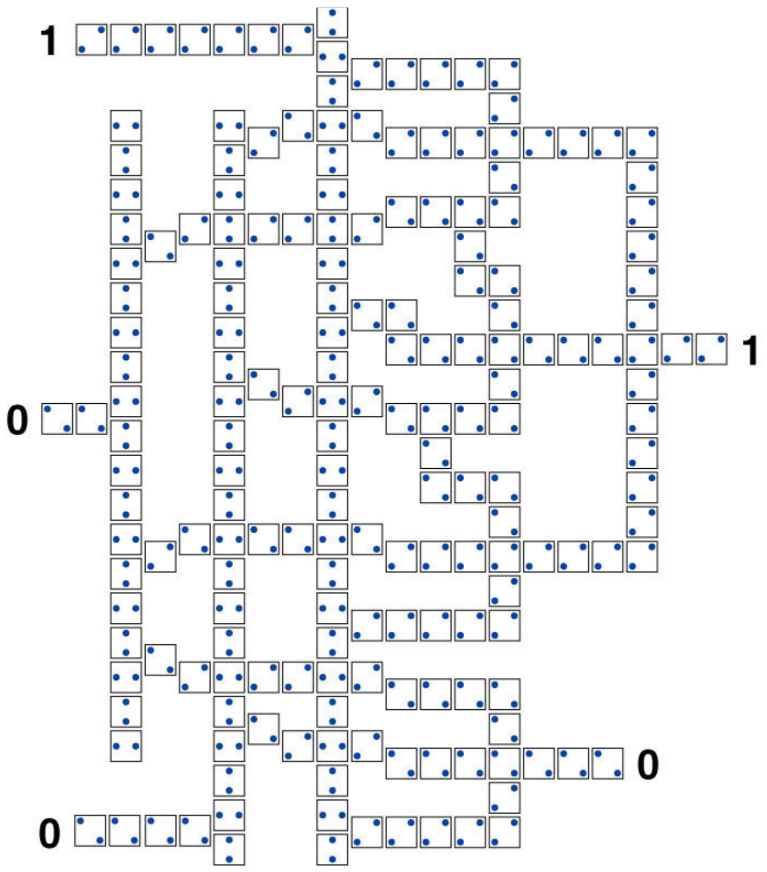


Figure 1.2: A leveled drawing of a QCA half-adder. (Courtesy of Greg Snider, University of Notre Dame.)

set of modules that can be placed in one 'layer'. In DNA mapping, small fragments of DNA must be ordered when only the overlap between the different fragments can be observed. In practice, the observed data may also contain errors, leading Waterman and Griggs (1986) to suggest solving a corresponding *maximum biplanar subgraph problem*.

In many of these cases, we need to represent the relationships graphically so that the location of the vertices are as consistent as possible with the transitivity of the relationship. That is, the edges should 'flow' in a uniform direction. Whether this direction is top-to-bottom or left-to-right is dependent upon the application domain. This necessitates a special type of graph drawing, called a *hierarchical* or *leveled drawing*.

1.1.1 Leveled Drawings and the Sugiyama Framework

Some graphs are labeled so as to partition the vertex set into layers or levels. For many others, the partitioning can be performed algorithmically. A drawing algorithm that handles this additional information seeks to assign the vertices of each level to a shared y-coordinate that is greater than that of the previous level. Algorithms for drawing such *leveled graphs* are typically based on the *Sugiyama framework* (Warfield, 1977; Sugiyama et al., 1981; Carpano, 1980; Eades and Sugiyama, 1991).

Briefly, the Sugiyama framework takes as input a (directed) graph and produces as output a leveled drawing of the graph by:

1. temporarily reversing a small or minimum number of edges so as to make the graph *acyclic* (using either heuristics or approximation algorithms (Berger and Shor, 1990; Eades et al., 1993; Sander, 1999; Eades and Lin, 1995; Saab, 2001; Flood, 1990; Demetrescu and Finocchi, 2003) or exact methods (Grötschel et al., 1985; Mitchell and Borchers, 1996));

2. finding a *leveling* of the graph (that minimizes the sum of the edge lengths (Gansner et al., 1993), minimizes or restricts the width (Coffman and Graham, 1972; Branke et al., 2002; Nikolov et al., 2005; An-

dreev et al., 2007), minimizes the height (Eades and Sugiyama, 1991), or all of the above (Healy and Nikolov, 2002; Nikolov, 2002));

3. ordering the vertices within each level so as to improve readability by

   (a) reducing or minimizing the number of edge crossings (see (Purchase, 1998) for the motivation and §4.1 for the methods), or

   (b) maximizing the size of a *level planar subgraph* (Mutzel, 2001a) (see also §4.1);

4. assigning the vertices horizontal coordinates (Sugiyama et al., 1981; Gansner et al., 1993; Sander, 1995; Buchheim et al., 2001; Brandes and Köpf, 2002);

5. restoring the direction of the reversed edges.

Researchers have also developed a host of alternatives and extensions to these steps (Newbery, 1989; Gansner et al., 1993; Bastert and Matuszewski, 1999; Mutzel, 2001b; Eiglsperger et al., 2004). Do Nascimento and Eades (2001) used 'user hints', particularly focus and manual changes, to help with the optimization process of the Sugiyama framework. Their pilot study showed that user interaction can help guide the algorithms for the steps above and, in some cases, avoid local minima.

The special case of drawing leveled graphs without any crossings has been investigated by Leipert (1998) and Kuusik (2000). Leipert (1998) developed an algorithm to test for this special case and, if the test succeeds, produce a leveled drawing without any crossings in linear running-time. We describe his work in more detail in §3.1. Bachmaier (2004) has also extended his approach to handle *radial level planarity* where the drawings can be wrapped around a vertical standing cylinder and *cyclic level planarity* where the drawings can be wrapped around a horizontal lying cylinder (see §3.3.1). Kuusik (2000) presented a conceptually simpler, albeit asymptotically slower, algorithm to test for this special case in quadratic running-time and, if the test succeeds, produce a leveled drawing without any crossings in cubic running-time. In doing so, he developed the notion of a (labeled-) vertex-exchange graph. We describe this graph in §2.4, show its connection to *signed graphs* and, in Chapter 3, build upon the work of Kuusik (2000) to eliminate the overhead of the cubic running-time when producing the leveled drawing without any crossings. We also extend his approach to handle a family of embedding constraints and tackle the more general case of drawing a leveled graph with the minimum number of crossings.

A multitude of tools can produce drawings of leveled graphs. In fact, in a recent survey (Jünger and Mutzel, 2004), thirteen of the fourteen graph drawing tools reviewed are based on toolkits that have the ability the produce drawings of hierarchical or leveled graphs.

1.1.2 Other Methods of Drawing Leveled Graphs

Leveled graphs have, by their very nature, a direction, *e.g.* top-to-bottom or left-to-right. Instead of stratifying the vertices onto discrete parallel lines, we can produce drawings where the edges adhere to this direction but the y-coordinates can assume any value. Some of the noteworthy approaches are described.

Force-Directed or Energy Minimization Methods

The force-directed method involves a gradient-descent minimization of an energy function based on some physical metaphor (see Brandes (2001) for a survey). For example, imagine that there are repulsive forces between vertices but attractive forces between the end points of edges. The closer the vertices are, the greater the magnitude of the repulsive forces whereas the further apart the end points of the edges are, the greater the magnitude of the attractive forces. The borders of the drawing must also repel the vertices to stop them from occupying too much area. Some of the vertices might have fixed positions while others have variable positions. If the variable vertices are initially 'pinned' to the drawing in a random set of positions, each vertex is acted upon by a set of forces. If the vertices are released, they accelerate according to these forces. The vertices come to rest only when the resultant forces acting on each provide enough deceleration to stop all movement. At this point, each edge is either relaxed or the repulsive forces in the system are preventing it from relaxing. Some discretized version of this system can be simulated by a computer. Many of the complexities of the real-world can be omitted; for example, if our vertices cannot rotate then we can ignore the moment of each force. Indeed, the force equations themselves can be defined to our liking. However, for a workable set of equations, the vertices should move less and less after each iteration until a point is reached when the average movement is less than some predefined value. The final position of the vertices provide the drawing. Force-directed methods have been augmented to handle edge directions, and can thus be used to draw directed graphs, *e.g.* see the modifications of Eades' spring embedder (Eades, 1984) by Sugiyama and Misue (1995). We provide more examples of force-directed and energy minimization methods in §2.6.

In contrast to the Sugiyama framework, the force-directed and energy minimization methods described in this section produce an 'organic' variety of drawing: clusters and symmetries, if they exist naturally, often emerge in the drawing and the proximity relations between the vertices are preserved. However, the Sugiyama framework may be preferred for more 'rigid' drawings, *e.g.* where the vertices are assigned to discrete grid coordinates. Criteria like the number of edge crossings or the size of the maximum level planar subgraph can also be explicitly optimized as part of the Sugiyama framework. This thesis is primarily focused on these steps.

1.2 Contributions and Organization

Chapter 2 introduces definitions and notation used throughout the thesis.

Chapter 3 is concerned with level planar (crossing-free) drawings. We extend the work of Healy and Kuusik (2004) to show how to determine the crossing-free orderings of the vertices in quadratic running-time and handle a family of *embedding constraints*. Healy and Kuusik (2004) describe a method of testing whether crossing-free orderings of the vertices of a level graph exist. However, if the test succeeds, their method cannot provide the actual crossing-free orderings of the vertices within the same asymptotic running-time. Instead, they require cubic running-time to produce the layout. We also show how the existing body of algorithms for the *maximum balanced subgraph problem* can be applied to the *multi-level crossing minimization problem*. The results of this chapter have been partially reported by Harrigan and Healy (2008a).

Chapter 4 considers the *multi-level crossing minimization* and *maximum level planar subgraph problems* for the special case of trees. We introduce the *planarizing penalty digraph* (a concept analogous to the *crossing penalty digraph*). We show that the maximum level biplanar subgraph problem can be reduced to the *Hamiltonian cycle completion problem*, for which there exist efficient algorithms for some special classes of graphs. Finally, we prove the NP-Hardness of the crossing minimization problem (for trees) when there are only two levels with the order of the vertices on one of the levels subject to some partial order and also for an arbitrary number of levels.

Chapter 5 presents a method of drawing a directed graph that emphasizes a *significant spanning tree*. The spanning tree is a *tree DAG* with multiple sources and so it is more appropriate to give each of these vertices root status than to nominate any single vertex and hang the subtrees from it. We ensure the *level planarity* of the spanning tree by inserting dummy vertices and restricting the possible leveled drawings so that no two significant edges cross. We also show that finding a minimum number of dummy vertices to ensure level planarity even when the tree DAG has exactly one vertex of in-degree greater than one is NP-Hard. The results of this chapter have been reported by Harrigan and Healy (2007, 2008b).

We conclude and present some open problems in Chapter 6.

Chapter 2

Preliminaries

In this chapter we provide some definitions and notation. We present the basics of graph theory, paying special attention to the topological property of *level planarity*. We also introduce the *(labeled) vertex-exchange graph* which is an example of a *signed graph* that has particular relevance to level planarity.

2.1 Graph Theory

We begin by covering undirected graphs, directed graphs and their matrix representations.

2.1.1 Undirected Graphs

Throughout this thesis, $G = (V, E)$ is a graph with vertex set $V(G)$ and edge set $E(G)$. A graph is *simple* if it contain no *loops* (an edge joining a vertex to itself) or *multiple edges* (more than one edge joining the same pair of vertices). We assume G is simple and undirected (the end points of each edge are unordered) unless explicitly stated otherwise. The number of vertices and edges in G are denoted by $n = |V(G)|$ and $m = |E(G)|$ respectively. Two vertices are *adjacent* if they are joined by an edge. A vertex and edge are *incident* if the vertex is one of the edge's end points. Two edges are *independent* if they do not share a vertex. The *degree* of a vertex v, $\texttt{deg}(v)$, is the number of edges incident to v. The set of *neighbors* of v, $\mathcal{N}(v)$, is the set of vertices adjacent to v.

A *weighted graph* associates a non-negative number (*weight*) with every edge in the graph.

The *complement* G^c of a graph G is a graph with the same vertex set as G but with an edge set such that $\{u, v\} \in E(G^c)$ if and only if $\{u, v\} \notin E(G)$.

A *walk* is an alternating sequence of vertices and edges, beginning and ending with a vertex, in which each vertex is incident to the two edges that precede and follow it in the sequence, and the vertices that precede and follow an edge are the end points of that edge. In a simple graph it can be specified simply by the sequence of vertices. The *length of a walk* is the number of edges it traverses. A *path* is a walk in which no vertices (and thus no edges) are repeated. Two paths are *disjoint* if they do not have any vertex in common, except possibly the first and last ones. A *cycle* is a path that ends on the vertex at which it started. A graph is *acyclic* if it does not contain any cycles. A *unicyclic graph* is a graph with exactly one cycle. A path or cycle is *Hamiltonian* if it includes all the vertices of the graph.

A *subgraph* of a graph G is a graph whose vertex and edge sets are subsets of those of G. A subgraph H is a *spanning subgraph* of a graph G if it has the same vertex set as G. We say H spans G. A subgraph H of a graph G is *induced* if, for any pair of vertices $u, v \in V(H)$, $\{u, v\} \in E(H)$ if and only if $\{u, v\} \in E(G)$.

If it is possible to establish a path from any vertex to any other vertex of a graph, the graph is said to be *connected*; otherwise, the graph is *disconnected*. A *cut vertex* is a vertex whose removal disconnects the graph. A *cut set* is a set of vertices whose removal disconnects the graph. A graph is *k-connected* if and only if it contains k disjoint paths between any two vertices. *Biconnected* is a synonym for 2-connected. A *component* is a connected subgraph that is maximal with respect to inclusion. Two vertices are in the same component if and only if there exists a path between them. In a drawing of a graph, the components can each be drawn separately with empty space between them. A nonempty connected graph has just one component. A component with just one vertex is a *trivial component*; otherwise it is *non-trivial*. A *clique* in a graph is a set of pairwise adjacent vertices.

A *forest* is an acyclic graph. A *tree* is a connected forest. A vertex of degree one in a tree is called a *leaf*. All other vertices are *inner vertices*. A *star* is a tree with at most one inner vertex. Sometimes, one vertex of the tree is distinguished, and called the *root*; in this case, the tree is called *rooted*, otherwise it is called a *free tree*. An inner vertex of a rooted tree has one or more *child vertices* and is called the *parent* of those children. In an *ordered tree* these children are ordered left-to-right. A *k-ary tree* is a rooted tree in which every inner vertex has k children. A 2-ary tree is also called a *binary tree*. A *complete k-ary tree* is an ordered k-ary tree in which all leaves are joined to the root by a path of length p or $p-1$, for some $p > 0$, and all inner vertices are as far left as possible. A *caterpillar* is a tree such that deleting its leaves gives a (possibly empty) path. This path is known as the *backbone* of the caterpillar. A *directed tree* is a rooted tree in which every edge from a parent to a child vertex is directed. A vertex v is a *descendant* of a vertex u in a directed tree if there is a directed path from u to v. A *subforest* (*subtree*) of a forest F (tree T) is a subgraph of F (T). A *spanning forest* (*spanning tree*) of a graph is a spanning subgraph that is itself a forest (tree). Every graph has spanning forests but only a connected graph has spanning trees.

2.1.2 Directed Graphs

A *directed edge* is an edge whose end points are ordered. The first vertex is called the *tail*; the second is called the *head*. A *directed graph*, or *digraph*, is analogous to an (undirected) graph except that it contains only directed edges. The *underlying undirected graph* of a directed graph is found by ignoring the edge directions.

In a directed graph G, we distinguish between the *in-degree*, $\texttt{inDeg}(v)$, the number of edges entering a vertex v, and the *out-degree*, $\texttt{outDeg}(v)$, the

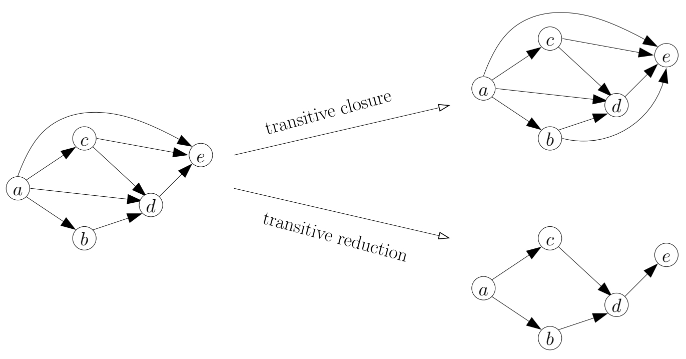


Figure 2.1: The transitive closure and transitive reduction of a directed graph.

number of edges leaving a vertex v. The degree of a vertex v is equal to the sum of its in- and out- degrees. A *source* is a vertex with zero in-degree; a *sink* is a vertex with zero out-degree. We also distinguish between the *outgoing* ($\mathcal{N}^+(v)$) and *incoming neighbors* ($\mathcal{N}^-(v)$) of a vertex v as the heads and tails of edges that leave and enter a vertex v respectively.

A *directed path*, or just a path when the context is clear, is a path whose directed edges all go in the same direction. A *directed cycle*, or just a cycle when the context is clear, is a cycle whose directed edges all go in the same direction. A *directed acyclic graph* or *DAG* is a directed graph with no directed cycles. However, its underlying undirected graph may contain cycles.

A *directed clique* in a graph is a set of vertices with an edge going in each direction between each pair of vertices (multiple edges). A *tournament* is a directed graph with a single edge going in one of the two directions between each pair of vertices. An *acyclic tournament* is a tournament that does not contain any directed cycles.

The *transitive closure* of a directed graph G is a directed graph G' with vertex set $V(G)$ and $(u, v) \in E(G')$ if and only if there is a directed path from u to v in G. A *transitive reduction* of a directed graph G is a directed graph G' with vertex set $V(G)$ and a minimal edge set such that there is a directed path from u to v in G' if and only if $(u, v) \in E(G)$. Figure 2.1 exemplifies both the transitive closure and transitive reduction of a directed graph. We use a transitive reduction in §5.4 to simplify a graph before drawing it.

2.1.3 Matrix Representations

The *adjacency matrix* $\mathcal{A}(G)$ of an undirected graph G is a square $n \times n$ matrix defined by

$$\mathcal{A}(G)_{u,v} = \begin{cases} 1 & \text{if } \{u, v\} \in E(G), \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

The *incidence matrix* $\mathcal{B}(G)$ of an undirected graph G is a rectangular $n \times m$ matrix defined by

$$\mathcal{B}(G)_{v,e} = \begin{cases} 1 & \text{if } v \text{ is incident to } e, \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

An *oriented incidence matrix* is an incidence matrix where each column has exactly one of its 1-entries negated. It represents an *orientation* of a graph: an assignment of exactly one direction to each of the graph's edges. The negated entry represents the tail and the positive entry represents the head. For example, suppose the incidence matrix of a graph G is given by

$$\mathcal{B}(T) = \begin{array}{c|cccccc|} & \{a,d\} & \{a,e\} & \{b,e\} & \{b,f\} & \{c,e\} & \{c,g\} \\ \hline a & 1 & 1 & 0 & 0 & 0 & 0 \\ b & 0 & 0 & 1 & 1 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & 1 & 1 \\ d & 1 & 0 & 0 & 0 & 0 & 0 \\ e & 0 & 1 & 1 & 0 & 1 & 0 \\ f & 0 & 0 & 0 & 1 & 0 & 0 \\ g & 0 & 0 & 0 & 0 & 0 & 1 \end{array},$$

then, an oriented incidence matrix of G is given by

$$\mathcal{B}(T) = \begin{array}{c|cccccc|} & \{a,d\} & \{a,e\} & \{b,e\} & \{b,f\} & \{c,e\} & \{c,g\} \\ a & 1 & 1 & 0 & 0 & 0 & 0 \\ b & 0 & 0 & 1 & 1 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & 1 & 1 \\ d & -1 & 0 & 0 & 0 & 0 & 0 \\ e & 0 & -1 & -1 & 0 & -1 & 0 \\ f & 0 & 0 & 0 & -1 & 0 & 0 \\ g & 0 & 0 & 0 & 0 & 0 & -1 \end{array}.$$

The *Laplacian* $\mathcal{L}(G)$ of an undirected graph G is a square $n \times n$ matrix defined by

$$\mathcal{L}(G)_{u,v} = \begin{cases} \mathtt{deg}(v) & \text{if } u = v, \\ -1 & \text{if } \{u,v\} \in E, \\ 0 & \text{otherwise.} \end{cases} \quad (2.3)$$

$\mathcal{L}(G)$ can also be computed from $\mathcal{L}(G) = \mathcal{B}(G)\mathcal{B}(G)_T$ where $\mathcal{B}(G)$ is any oriented incidence matrix of G. $\mathcal{L}(G)$ is positive semi-definite and so its eigenvalues, $\lambda_1 \leq \ldots \leq \lambda_n$, are all nonnegative. An eigenvector corresponding to the second smallest eigenvalue is known as a *Fiedler vector* and can be calculated using power iteration (Hotelling, 1933) on the matrix $\lambda_n I - \mathcal{L}(G)$ or more efficiently using multi-scale methods (Harel and Koren, 2002).

2.2 Orders

A *strict partial order* is a binary relation $\prec$ over a set S which is irreflexive, asymmetric, and transitive, *i.e.* $\forall a, b, c \in S$: $\neg(a \prec a)$ (irreflexive), if $a \prec b$ then $\neg(b \prec a)$ (asymmetric) and if $a \prec b$ and $b \prec c$ then $a \prec c$ (transitive). The strict qualifier is assumed from now on. Every partial order corresponds to a DAG with vertex set S and vice versa, *i.e.* $a \prec b$ if and only if there is a directed path from a to b in G.

A *total order* is a partial order that is also *total*: $\forall a, b \in S$, either $a \prec b$ or $b \prec a$. A total order T is a *linear extension* of a partial order P if, whenever $a \prec b$ in P then $a \prec b$ in T. A linear extension of a partial order corresponding to a DAG is a *topological order* of the DAG's vertices.

2.3 Levelings and Level Planarity

A *leveling* of a graph G is a surjective mapping $\phi : V(G) \to \{1, 2, \ldots, k\}$ such that $\phi(u) \neq \phi(v), \forall \{u, v\} \in E(G)$. A *leveled graph* is simply a graph with a leveling. A leveling is *proper* if $\forall \{u, v\} \in E(G) : |\phi(u) - \phi(v)| = 1$. In the following we assume all levelings to be proper. A *proper leveling* partitions the vertex set $V(G) = V_1 \cup V_2 \cup \ldots \cup V_k$ such that $V_i = \phi^{-1}(i)$ and the edge set $E(G) = E_1 \cup E_2 \cup \ldots \cup E_{k-1}$ such that $E_i \subseteq V_i \times V_{i+1}$.

A *hierarchy* is a leveled graph G in which every vertex $v \in V_j$ for $j > 1$ has at least one incident edge $\{u, v\} \in E(G)$ with $u \in V_{j-1}$.

A *leveled embedding* or *k-leveled embedding* of G with leveling ϕ consists of total orders $\prec_j$ of the vertices in $\phi^{-1}(j), 1 \leq j \leq k$. When the level in question can be determined from the context, we simply use $\prec$. A *leveled drawing* or *k-leveled drawing* of G with leveling ϕ is a drawing in which the vertices in $\phi^{-1}(j), 1 \leq j \leq k$, are placed on horizontal levels l_j and the edges are drawn as straight-line segments between the vertices. Every k-leveled drawing induces a k-leveled embedding. In a leveled drawing of a hierarchy, every vertex is reachable via a *ϕ-monotonic path* from a source at the top level. A path in a leveled graph is ϕ-monotonic if, for all $u, v \in V(G)$ where u precedes v in the sequence of vertices in the path, $\phi(u) < \phi(v)$. Figure 2.2 exemplifies a k-leveled drawing of a graph G with leveling ϕ. From the drawing we can see that, for example, $u \prec_3 w$, or simply $u \prec w$.

We can also note the following two observations:

Observation 2.3.1. *For a graph G with leveling ϕ and a k-leveled drawing of G, two edges $\{u, y\}, \{v, x\} \in E(G)$, where $u, v \in \phi^{-1}(j)$ and $x, y \in \phi^{-1}(j+1)$, cross if and only if $u \prec v$ and $x \prec y$, or $v \prec u$ and $y \prec x$.*

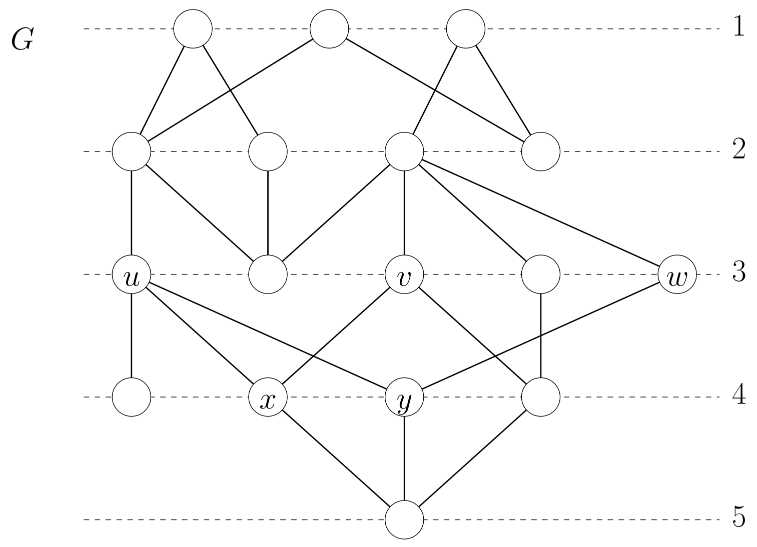


Figure 2.2: A k-leveled drawing of a graph G with leveling ϕ.

Observation 2.3.2. *For a graph G with leveling ϕ, a k-leveled drawing of G with the vertices on one level, say level j, fixed and two vertices $x, y \in \phi^{-1}(j+1)$ on an adjacent level free to move anywhere along that level, the number of crossings between edges incident with x and edges incident with y in between levels j and $j+1$ is completely determined by the relative order of x and y,* i.e. *whether $x \prec y$ or $y \prec x$.*

A k-leveled drawing of G with leveling ϕ is *level planar* if no two edges cross except at common end points. G with leveling ϕ is level planar if and only if it has a k-leveled drawing that is level planar. A k-leveled embedding is level planar if and only if every k-leveled drawing of G in which the order of the vertices along l_j satisfy $\prec_j, 1 \leq j \leq k$, is level planar.

More formally, this can be stated as follows:

Definition 2.3.3. *A graph G with leveling ϕ is level planar if and only if there exists a leveled embedding $\prec$ of G such that:*

1. *For every pair of* independent edges *in G between the same two levels, $\{u, y\}, \{v, x\} \in E(G)$ where $u, v \in \phi^{-1}(j)$ and $x, y \in \phi^{-1}(j+1)$, $(u \prec v \wedge y \prec x) \vee (v \prec u \wedge x \prec y)$,* i.e. *the edges do not cross.*

2. *For every three vertices on the same level, $u, v, w \in \phi^{-1}(j)$, $(u \prec v \wedge v \prec w) \implies u \prec w$,* i.e. *transitivity holds.*

The number of edge crossings in a k-leveled drawing is the number of pairs of edges that cross, *i.e.* the number of pairs of independent edges that violate Condition 1. To make this evident from the drawing, we require that no point between the same two levels lie on more than two edges.

A 2-leveled graph must be *bipartite*, *i.e.* it cannot contain a cycle of odd length. A 2-leveled graph that is level planar is said to be *biplanar*.

Finally, in considering level planar graphs, we note the following bound on the number of edges m:

Lemma 2.3.4. *For a connected level planar graph G with leveling ϕ and $n > 2$, $m \leq 2n - 4$.*

Proof: Let G have k levels.

Case $k = 1$: G has no edges, inequality is trivially satisfied.

Case $k = 2$: G is a tree, so $m \leq n - 1$ and since $n > 2$, we have $m \leq 2n - 4$.

Case $k > 2$: n is the number of vertices on all k levels, $n = \sum_{i=1}^{k} |\phi^{-1}(i)|$. Between any pair of consecutive levels i and $i + 1$ there are at most $|\phi^{-1}(i)| + |\phi^{-1}(i+1)| - 1$ edges (by the previous case). Therefore, summing over all pairs of consecutive levels,

$$\begin{aligned} m &\leq |\phi^{-1}(1)| + 2\sum_{i=2}^{k-1} |\phi^{-1}(i)| + |\phi^{-1}(k)| - k + 1 \\ &\leq 2\sum_{i=1}^{k} |\phi^{-1}(i)| - |\phi^{-1}(1)| - |\phi^{-1}(k)| - k + 1 \\ &\leq 2n - 4. \end{aligned}$$

□

Therefore, any graph G with leveling ϕ that is level planar has $m = \mathcal{O}(n)$ edges. Of course, since level planarity implies planarity (but not vice versa), this asymptotic bound can also be deduced from Euler's formula (Euler, 1758).

2.4 Signed Graphs and the Vertex-Exchange Graph

A *signed graph* is a graph G with a *sign mapping* or *signature* $\lambda : E(G) \rightarrow \{`+', `-'\}$. It was introduced by Cartwright and Harary (1956) to model a problem in social psychology. For a comprehensive bibliography of signed graphs and their varied uses, see the survey of Zaslavsky (1999).

Any vertex $v \in V(G)$ of a signed graph can be *switched*. This involves switching the sign or labeling of every edge incident with v, *i.e.* '–'-labels become '+'-labels and '+'-labels become '–'-labels. Harary (1953–1954) formulated the notion of *balance* with respect to signed graphs: can we perform some sequence of switches that removes all the '–'-labels? Alternatively, G with signature λ is balanced if and only if the set of '–'-labeled edges form a *cut*. A cut is a partition of $V(G)$ into two sets V_1 and V_2 such that every edge $\{u, v\} \in E(G)$ with $u \in V_1$ and $v \in V_2$ is '–'-labeled (the cut edges) and every other edge is '+'-labeled. We can test if G with signature λ is balanced in $\mathcal{O}(n + m)$ running-time (Harary and Kabell, 1980; Loukakis, 2003).

The *vertex-exchange* and *labeled vertex-exchange graphs* were first introduced by Healy and Kuusik (2004).

Definition 2.4.1. *The* vertex-exchange graph $\mathcal{VE}(G, \phi)$ *of a graph G with leveling ϕ is a graph with vertex set $V(\mathcal{VE}(G, \phi)) = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \ldots \cup \mathcal{V}_k$ where $\mathcal{V}_j = \{\langle u, v\rangle | u, v \in V_j, u \neq v\}$ and edge set $E(\mathcal{VE}(G, \phi)) = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_{k-1}$ where $\mathcal{E}_j = \{\{\langle u, v\rangle, \langle x, y\rangle\} | \{u, y\}, \{v, x\} \in E_j, \langle u, v\rangle \in \mathcal{V}_j, \langle x, y\rangle \in \mathcal{V}_{j+1}\}$.*

In other words, $\mathcal{VE}(G, \phi)$ is constructed by taking the distinct pairs of vertices on the same level of G as vertices of $\mathcal{VE}(G, \phi)$ and joining two vertices in $\mathcal{VE}(G, \phi)$ whenever two pairs from the four corresponding vertices in G are joined by independent edges (see Fig. 2.3). $\langle u, v\rangle$ denotes an unordered pair of same-level vertices u and v. Note that $\langle u, v\rangle \equiv \langle v, u\rangle$.

Healy and Kuusik (2004) augment the vertex-exchange graph $\mathcal{VE}(G, \phi)$ with an edge labeling $\lambda : E(\mathcal{VE}(G, \phi)) \rightarrow \{`+', `-'\}$ to produce the *labeled* *vertex-exchange graph* $\mathcal{LVE}(G, \phi, \mathcal{L})$ as follows. Choose some initial leveled embedding $\mathcal{L}$ of G. For every edge $e \in E(\mathcal{VE}(G, \phi))$ they set $\lambda(e) =$ '$-$' if the corresponding edges in G cross and $\lambda(e) =$ '$+$' if they do not (see Fig. 2.3).

The following properties of the labeled vertex-exchange graph are apparent. A path P in $\mathcal{LVE}(G, \phi)$ induces a pair of walks in G. These walks are not necessarily vertex or edge disjoint. A degree-one vertex in $\mathcal{LVE}(G, \phi)$ corresponds to two degree-one vertices in G. $\mathcal{LVE}(G, \phi)$ may contain biconnected components even when G is a tree. $\mathcal{LVE}(G, \phi)$ may be disconnected even when G is connected.

There is one proviso with these definitions. If G contains a $K_{2,2}$ subgraph then the corresponding vertex-exchange graph will no longer be simple; it will be a graph with multiple edges. In the labeled vertex-exchange graph, one of the edges contributed by the $K_{2,2}$ will be labeled '$+$' and the other '$-$'. This detail can be handled by the algorithms in the next chapter.

Of course, a labeled vertex-exchange graph is an example of a signed graph. Performing a switch on a vertex of a labeled vertex-exchange graph changes the labels of all the incident edges. The switch loosely corresponds to exchanging the position of two vertices in the leveled embedding $\mathcal{L}$ of G. We shall study this correspondence more closely in §3.2.2.

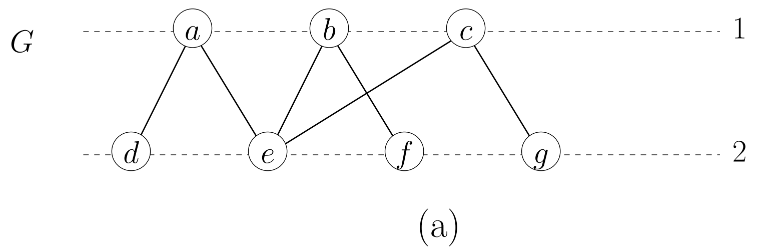
The relevance of the labeled vertex-exchange graph to level planarity was first established by Healy and Kuusik (2004). We now state their theorem but defer our alternative proof until §3.2.

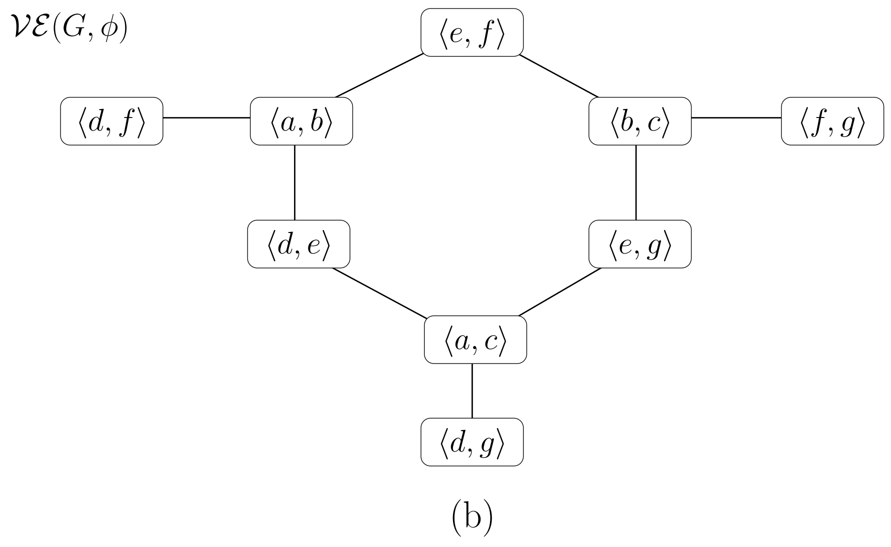
Theorem 2.4.2. *(Healy and Kuusik, 2004)*

A graph G with leveling ϕ is level planar if and only if its labeled vertex-exchange graph $\mathcal{LVE}(G, \phi, \mathcal{L})$ with respect to any initial leveled embedding $\mathcal{L}$ of G is balanced.

2.5 The Parameterized Line Graph

The vertex-exchange graph $\mathcal{VE}(G, \phi)$ of a graph G with leveling ϕ can easily be constructed from the definitions in §2.4. However, in this section we briefly





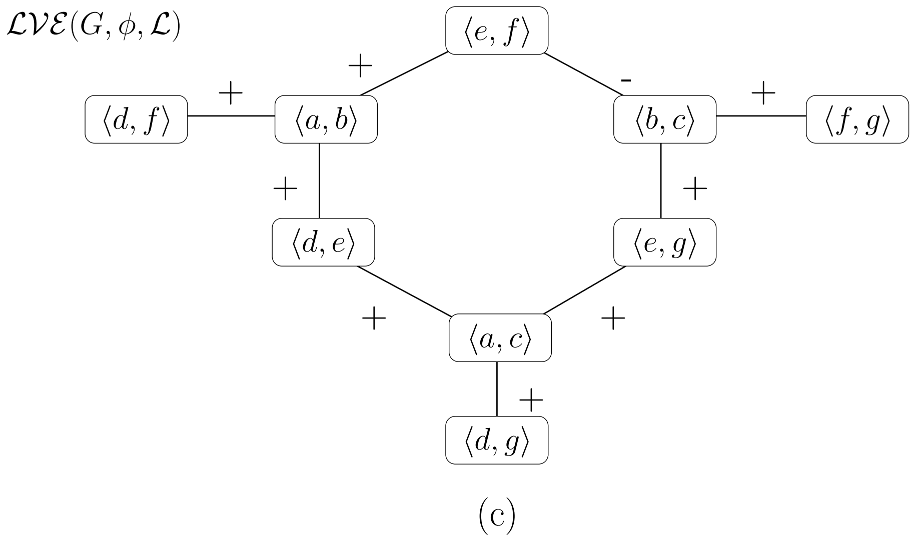


Figure 2.3: A (a) leveled graph, its (b) vertex-exchange graph and (c) labeled vertex-exchange graph.

show how, for the case of G being a tree and ϕ having exactly two levels, it can also be constructed using a *parameterized line graph*. This construction is not used in the following chapters, but is somewhat interesting in and of itself.

The *line graph* $\mathcal{LG}(G)$ of a graph G is constructed by taking the edges of G as vertices of $\mathcal{LG}(G)$ and joining two vertices in $\mathcal{LG}(G)$ whenever the corresponding edges in G have a common vertex. The adjacency matrix of $\mathcal{LG}(G)$ can be computed as follows:

$$\mathcal{A}(\mathcal{LG}(G)) = \mathcal{B}(G)^T \cdot \mathcal{B}(G) - 2I \tag{2.4}$$

where $\mathcal{A}(G)$ is the adjacency matrix of G, $\mathcal{B}(G)$ is an oriented incidence matrix of G, M^T is the transpose of a matrix M and I is the identity matrix. Subtracting $2I$, from the right hand side of Eq. 2.4 removes the pair of loops from each vertex in $\mathcal{LG}(G)$ (since every edge shares two vertices with itself).

The *parameterized line graph* $\mathcal{LG}(G, H)$ of two graphs G and H, both having the same vertex set, is constructed by taking the edges of G as vertices of $\mathcal{LG}(G, H)$ and joining two vertices in $\mathcal{LG}(G, H)$ whenever at least two of the incident vertices of the corresponding edges in G are adjacent in H. The adjacency matrix of $\mathcal{LG}(G, H)$ can be computed as follows:

$$\mathcal{A}(\mathcal{LG}(G, H)) = \mathcal{B}(G)^T \cdot \mathcal{A}(H) \cdot \mathcal{B}(G) \tag{2.5}$$

where the order of the vertices represented by the rows and columns of $\mathcal{A}(H)$ must match the order of those represented by the rows of $\mathcal{B}(G)$. Note that $\mathcal{LG}(G) = \mathcal{LG}(G, I) - 2I$.

The parameterized line graph $\mathcal{LG}(G, G^c)$ of a graph G is constructed by taking the edges of G as vertices of $\mathcal{LG}(G, G^c)$ and joining two vertices in $\mathcal{LG}(G, G^c)$ whenever at least two of the incident vertices of the corresponding edges in G are not adjacent. Note that $\mathcal{LG}(G, G^c)$ is distinct from $\mathcal{LG}(G)^c$ and $\mathcal{LG}(G^c)$.

As one last variation on this idea, we construct $\mathcal{LG}(G, \texttt{even})$ by taking the edges of G as vertices of $\mathcal{LG}(G, \texttt{even})$ and joining two vertices in $\mathcal{LG}(G, \texttt{even})$

whenever the length of the shortest path between the corresponding edges (from either of the first edge's endpoints to either of the second edge's endpoints) in G is even. $\mathcal{LG}(G, \texttt{odd})$ can be constructed analogously. Note that paths of length zero are considered even, so $\mathcal{LG}(G)$ is a subgraph of $\mathcal{LG}(G, \texttt{even})$.

Now, suppose we are given a tree T with leveling ϕ over exactly two levels. Consider the following parameterized line graph:

$$\mathcal{LG}(\mathcal{LG}(T), \mathcal{LG}(T)^c) - 2I \tag{2.6}$$

where, in this instance, $\mathcal{LG}(T)^c$ also contains a loop on each vertex. There is an edge in $\mathcal{LG}(T)$ for every pair of edges in T that share a vertex. Therefore, there is a vertex in the parameterized line graph for every pair of edges in T that share a vertex. But every such pair of edges in T that share a vertex on one level must have, as their opposite end points, two disjoint vertices on the other level. These two vertices contribute a single vertex to $\mathcal{VE}(T, \phi)$. Since T is a tree and cannot contain any $K_{2,2}$ subgraph, we have an injective mapping from the vertices of the parameterized line graph to the vertices of $\mathcal{VE}(T, \phi)$.

Using this mapping between the vertices of the parameterized line graph and the vertices of the vertex-exchange graph, we can also see a correspondence between some of the edges. Suppose there exists an entry in the adjacency matrix of the parameterized line graph with an absolute value of 2, *i.e.* $|\mathcal{A}(X)_{u,v}| = 2$ where X is the parameterized line graph above. Then, there exists a pair of edges in $\mathcal{LG}(T)$ whose corresponding vertices are joined by exactly two edges in $\mathcal{LG}(T)^c$. This can only happen if the pair of edges in $\mathcal{LG}(T)$ share exactly one vertex: the first edge in $\mathcal{LG}(T)^c$ is the loop on the shared vertex and the second edge in $\mathcal{LG}(T)^c$ is the edge joining the other two end points of the edges in $\mathcal{LG}(T)$. However, a pair of edges sharing a vertex in $\mathcal{LG}(T)$ could only arise from a path of length three in T: $\{a, b\}, \{b, c\}, \{c, d\}$ where a and c are on one level and b and d are on the other. From the injective mapping above, u is mapped to one of $\langle a, c\rangle$ or $\langle b, d\rangle$ and v is mapped to the other. Since $\{a, b\}$ and $\{c, d\}$ are two independent edges joining a pair of vertices from one level to a pair of vertices on another level, $\{u, v\}$ is an edge in $\mathcal{VE}(T, \phi)$. However, the converse is not necessarily true: if $\{u, v\}$ is an edge in $\mathcal{VE}(T, \phi)$ then the corresponding entry in the adjacency matrix of the parameterized line graph may not have an absolute value of two.

In summary, by computing $\mathcal{LG}(\mathcal{LG}(T), \mathcal{LG}(T)^c) - 2I$ we can construct a subgraph of $\mathcal{VE}(T, \phi)$. As an example, consider again the leveled tree in Fig. 2.3. The adjacency matrix and an oriented incidence matrix of T are

$$\mathcal{A}(T) = \begin{array}{c|ccccccc|} & a & b & c & d & e & f & g \\ \hline a & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ c & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ d & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ e & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ f & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ g & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

and

$$\mathcal{B}(T) = \begin{array}{c|cccccc|} & \{a,d\} & \{a,e\} & \{b,e\} & \{b,f\} & \{c,e\} & \{c,g\} \\ \hline a & 1 & 1 & 0 & 0 & 0 & 0 \\ b & 0 & 0 & 1 & 1 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & 1 & 1 \\ d & -1 & 0 & 0 & 0 & 0 & 0 \\ e & 0 & -1 & -1 & 0 & -1 & 0 \\ f & 0 & 0 & 0 & -1 & 0 & 0 \\ g & 0 & 0 & 0 & 0 & 0 & -1 \end{array}$$

respectively. So,

$$\begin{aligned}
\mathcal{A}(\mathcal{LG}(T)) &= \mathcal{B}(T)^T \cdot \mathcal{B}(T) - 2I \\
&= \begin{array}{c|cccccc|}
 & \{a,d\} & \{a,e\} & \{b,e\} & \{b,f\} & \{c,e\} & \{c,g\} \\
\hline
\{a,d\} & 0 & 1 & 0 & 0 & 0 & 0 \\
\{a,e\} & 1 & 0 & 1 & 0 & 1 & 0 \\
\{b,e\} & 0 & 1 & 0 & 1 & 1 & 0 \\
\{b,f\} & 0 & 0 & 1 & 0 & 0 & 0 \\
\{c,e\} & 0 & 1 & 1 & 0 & 0 & 1 \\
\{c,g\} & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
\end{aligned}$$

and

$$\mathcal{A}(\mathcal{LG}(T)^c) = \begin{array}{c|cccccc|}
 & \{a,d\} & \{a,e\} & \{b,e\} & \{b,f\} & \{c,e\} & \{c,g\} \\
\hline
\{a,d\} & 1 & 0 & 1 & 1 & 1 & 1 \\
\{a,e\} & 0 & 1 & 0 & 1 & 0 & 1 \\
\{b,e\} & 1 & 0 & 1 & 0 & 0 & 1 \\
\{b,f\} & 1 & 1 & 0 & 1 & 1 & 1 \\
\{c,e\} & 1 & 0 & 0 & 1 & 1 & 0 \\
\{c,g\} & 1 & 1 & 1 & 1 & 0 & 1
\end{array}.$$

Finally, the adjacency matrix of $\mathcal{LG}(\mathcal{LG}(T), \mathcal{LG}(T)^c) - 2I$ is computed as follows:

$$\mathcal{B}(\mathcal{LG}(T))^T \cdot \mathcal{A}(\mathcal{LG}(T)^c) \cdot \mathcal{B}(\mathcal{LG}(T)) - 2I = \begin{bmatrix} 0 & -2 & 1 & 0 & -2 & 1 \\ -2 & 0 & -2 & -1 & 1 & 0 \\ 1 & -2 & 0 & 2 & 0 & -1 \\ 0 & -1 & 2 & 0 & 1 & -2 \\ -2 & 1 & 0 & 1 & 0 & -2 \\ 1 & 0 & -1 & -2 & -2 & 0 \end{bmatrix}.$$

The six vertices (the rows and columns of the adjacency matrix above) are mapped to the six vertices in the cycle of the vertex-exchange graph (see Fig. 2.3) and the entries with an absolute value of two (ignoring symmetry) correspond to the six edges in the cycle. However, the other three vertices and three edges are not accounted for. The omitted vertices are those vertices whose corresponding pairs of vertices in the leveled graph do not share a vertex on the opposite level. To include these we need to consider the following parameterized line graph:

$$\mathcal{LG}(\mathcal{LG}(T, \texttt{even}), \mathcal{LG}(T, \texttt{even})^c) - 2I \tag{2.7}$$

where, in this instance, $\mathcal{LG}(T, \texttt{even})^c$ also contains a loop on each vertex. There is an edge in $\mathcal{LG}(T, \texttt{even})$, and hence a vertex in the parameterized line graph, for every pair of edges in T. Every pair of edges in T either share a vertex on one level and must have, as their opposite end points, two disjoint vertices on the other level or, are independent and have two endpoints that are furthest away from each other. These two vertices, in either case, contribute a single vertex to $\mathcal{VE}(T, \phi)$. Since T is a tree and cannot contain any $K_{2,2}$ subgraph, we again have an injective mapping from the vertices of the parameterized line graph to the vertices of $\mathcal{VE}(T, \phi)$. This time all the vertices of the vertex-exchange graph are included. We also get the same correspondence between the edges, *i.e.* any entry in the adjacency matrix of the parameterized line graph with an absolute value of two represents an edge in the vertex-exchange graph. The converse is also true.

This construction is not intended for use in the algorithms of the following chapters. It is quite limited, in that it only constructs the vertex-exchange graph for trees on two levels. Instead, it is intended as a first step towards constructing vertex-exchange graphs for more general cases using parameterized line graphs. The more general idea of parameterizing a line graph with adjacency matrices may also be useful in network analysis (Brandes and Erlebach, 2005). Line graphs are a useful tool when computing edge betweenness and identifying clusters (Newman and Girvan, 2004). The ability to parameterize these line graphs with another (possibly disparate) source of information needs investigation.

2.6 Force-Directed and Energy Minimization Methods

In this section we return to the force-directed and energy minimization methods of the previous chapter. In particular, we look at approaches that involve the separation of axes.

Carmel et al. (2002, 2004) propose separating the x- and y-axes and applying an energy minimization strategy to each. Given a weighted directed graph G, they determine distinct coordinates for each vertex in 2-dimensional space as follows.

For the y-axis, they associate a *target height difference*, $\delta(u, v)$, with each directed edge $(u, v) \in E(G)$. In the final drawing, they wish to place u and v such that $y_u - y_v = \delta(u, v) = -\delta(v, u)$ where $Y(G) = (y_{v_1}, \dots, y_{v_n})^T$ is a vector representing the coordinates of each vertex along the y-axis. If G' is the *underlying undirected graph* of G, then the problem can be formulated as

$$\min \sum_{\{u,v\} \in E(G')} w(u, v)(y_u - y_v - \delta(u, v))^2 \tag{2.8}$$

where $w(u, v)$ is the weight of the undirected edge $\{u, v\}$. Non-zero $\delta(u, v)$s prevent the collapse of all vertices to the same location. The *balance of a vertex* $u \in V(G)$ is defined by

$$b_u = \sum_{v:\{u,v\} \in E(G')} w(u, v)\delta(u, v). \tag{2.9}$$

It is the algebraic sum of the weighted target height differences of all the edges to which a vertex is incident. The *balance of a graph* G is defined by $B(G) = (b_{v_1}, \ldots, b_{v_n})^T$. It comprises the balances of all the vertices in the graph. We would like to find a position for each vertex so that the weighted target height differences are satisfied as closely as possible. Carmel et al. (2002, 2004) prove that a minimizer of Eq. 2.8 is a solution to the system of linear equations

$$\mathcal{L}(G')Y(G) = B(G) \tag{2.10}$$

where $\mathcal{L}(G')$ is the *Laplacian* of G'. A weighted symmetric version of $\mathcal{L}(G')$ has also been discussed by Brandes et al. (2001). To define a unique solution, they require that the center of mass of the arrangement be at the origin, *i.e.* $\sum_{u \in V(G)} y_u = 0$.

To determine the coordinates along the x-axis, they consider two alternatives:

- minimize the sum of the squared edge lengths; or
- minimize the sum of the edge lengths.

The former can be solved efficiently using a multi-scale algorithm known as ACE (Harel and Koren, 2002; Koren et al., 2003). The latter is the *minimum linear arrangement problem* and can be formulated as

$$\min \sum_{\{u,v\} \in E(G')} w(u, v)|x_u - x_v| \tag{2.11}$$

where $X(G) = (x_{v_1}, \ldots, x_{v_n})^T$ is a vector representing the coordinates of each vertex along the x-axis. This is an NP-Hard problem. However, another multi-scale algorithm (Koren and Harel, 2002) efficiently finds a locally optimal solution. As a final 'beautification' step, they employ the one-dimensional force directed model of Fruchterman and Reingold (1991) to improve the drawing.

This approach works very well for regular graphs (each vertex having the same degree) (Carmel et al., 2002, 2004). However, so-called *symmetric vertices* within cycles can lead to poor drawings. To overcome these difficulties, a drawing can be computed by minimizing a *stress* or *energy function* while imposing *hierarchy constraints* derived from Eq. 2.8 to keep the directionality of the graph (Gansner et al., 2004; Dwyer and Koren, 2005; Dwyer et al., 2006a,b).

Brandes and Cornelsen (2003) also separate the axes when visualizing the link structure and ranking of resources on the World Wide Web. They show the results of the ranking algorithm in one dimension and use graph drawing techniques in the remaining one or two dimensions to show the underlying structure.

Chapter 3

Level Planarity

In this chapter we study the relevance of the labeled vertex-exchange graph to testing a graph for level planarity and producing a level planar embedding if the test succeeds. In particular, we extend a method of level planarity testing (Healy and Kuusik, 2004) to produce a level planar embedding within the same asymptotic running-time (§3.2). We further show how to handle cyclic level planarity testing and embedding (§3.3.1), preserve embedding constraints that restrict the order of incident edges around certain vertices (§3.3.2) and tackle the k-level crossing minimization problem (§3.3.3) using fixed parameter tractable (FPT) and approximation algorithms. The results of this chapter have been reported previously (Harrigan and Healy, 2008a).

3.1 Previous Work

There exists a level planarity testing and embedding algorithm with linear running-time (Di Battista and Nardelli, 1988; Heath and Pemmaraju, 1996; Jünger and Leipert, 2002). The algorithm employs the *PQ-tree* data structure (Booth and Lueker, 1976). PQ-trees are useful for representing a family of permutations on a set of elements. The level planarity testing algorithm visits the vertices of a graph G with leveling ϕ level-by-level from 1 to k, updating a set of PQ-trees at each level. If an update does not result in a valid PQ-tree for some level then the leveled graph is not level planar.

A naïve approach to a level planarity embedding algorithm would be to visit the vertices of G bottom up from level k to level 1 and, for each level, choose any one of the valid vertex permutations allowed by the PQ-trees. Unfortunately, this process requires maintaining all intermediate PQ-trees from the level planarity testing process. Also, deciding positions for end vertices of *long edges* (edges with many dummy vertices) would involve timely traversals back along these edges and lead to $\mathcal{O}((n+m)^2)$ running-time, as observed by Jünger and Leipert (2002). Instead, they augment G to a planar *st-graph* such that its *st-numbering* follows the vertex ordering of a level planar embedding. This consists of adding a source s and a sink t to G along with a special set of edges connecting them to the rest of G and numbering the vertices $1, \ldots, n$ such that vertices 1 (s) and n (t) are adjacent and every other vertex y is adjacent to two vertices x and z such that $x \leq y \leq z$. A planar embedding of this *st*-graph is then calculated by the algorithm of Chiba et al. (1985) and a level planar embedding is obtained by a simple depth first search traversal. The most involved part of this process is augmenting G to a planar *st*-graph that preserves level planarity.

This approach to level planarity testing and embedding can also handle graphs with non-proper levelings within the same asymptotic running-time. However, it is a very complicated algorithm and has proved difficult to implement in practice. Our method in §3.2, while requiring quadratic running-time, is much simpler to understand and implement. It extends the work of Healy and Kuusik (2004) to show how to determine the crossing-free orderings of the vertices of a level planar graph in quadratic running-time and handle a family of *embedding constraints*. Healy and Kuusik (2004) describe a method of testing whether crossing-free orderings of the vertices of a level graph exist. They introduced the (labeled-) vertex-exchange graph and proved Theorem 3.2.1. However, if the level planarity test succeeds, their method cannot provide the actual crossing-free orderings of the vertices within the same asymptotic running-time. Instead, they provide a book-keeping solution that requires cubic running-time to produce the layout. Our extension produces that layout within quadratic running-time and allows for embedding constraints. We have also shown a connection between the (labeled-) vertex-exchange graph and signed graphs and used this to apply the existing body of algorithms for the *maximum balanced subgraph problem* to the *multi-level crossing minimization problem.*

3.2 Testing and Embedding

The following is an alternative statement of a theorem, due to Healy and Kuusik (2004), that shows the relevance of the labeled vertex-exchange graph to level planarity:

Theorem 3.2.1. *(Healy and Kuusik, 2004)*

A graph G with leveling ϕ is level planar if and only if its labeled vertex-exchange graph $\mathcal{LVE}(G, \phi, \mathcal{L})$ with respect to any initial leveled embedding $\mathcal{L}$ of G is balanced.

Proof: We describe an alternative proof for this theorem that uses some previously presented ideas (Healy and Kuusik, 2004; Randerath et al., 2001). It provides some intuition for the algorithms in the subsequent sections.

Necessity: If G with leveling ϕ is level planar then there exists some leveled embedding $\mathcal{L}'$ of G with no crossings. $\mathcal{LVE}(G, \phi, \mathcal{L}')$ must contain no '$-$'-labeled edges and is therefore balanced.

Sufficiency: $\mathcal{LVE}(G, \phi, \mathcal{L})$ with respect to some initial leveled embedding $\mathcal{L}$ of G is balanced if and only if the set of '$-$'-labeled edges form a cut. From this cut we can partition $V(\mathcal{LVE}(G, \phi, \mathcal{L}))$ into two disjoint subsets $\mathcal{V}_\alpha, \mathcal{V}_\beta$ in such a way that each '$+$'-labeled edge joins two vertices of the same subset and each '$-$'-labeled edge joins two vertices of different subsets. We now show that given such a partition, Conditions 1 and 2 from Definition 2.3.3 follow.

Let C be the number of *non-trivial* components in $\mathcal{LVE}(G, \phi, \mathcal{L})$. We have two cases:

Case $C = 1$: We first show that Condition 1 follows. The initial leveled embedding $\mathcal{L}$ of G provides a relative order for each pair of vertices in G represented by each vertex in $\mathcal{LVE}(G, \phi, \mathcal{L})$. Every edge in $\mathcal{LVE}(G, \phi, \mathcal{L})$ represents a pair of independent edges in G between the same two levels, $\{u, w\}, \{v, x\} \in E(G)$. If the edge is '$+$'-labeled then $(u \prec v \wedge w \prec x) \vee (v \prec u \wedge x \prec w)$. However, if the edge is '$-$'-labeled then the opposite is true, *i.e.* $(u \prec v \wedge x \prec w) \vee (v \prec u \wedge w \prec x)$. For each vertex $\langle u, v\rangle \in \mathcal{V}_\alpha$, we leave the relative order unchanged. For each vertex $\langle w, x\rangle \in \mathcal{V}_\beta$, we reverse the relative order. Now, irrespective of whether an edge was '$+$'-labeled or '$-$'-labeled, its corresponding vertices in G must satisfy $(u \prec v \wedge w \prec x) \vee (v \prec u \wedge x \prec w)$ and so Condition 1 must follow.

We now show that Condition 2 follows. We use the relative orders of the vertices from the previous paragraph. Suppose there exists three vertices on the same level, $u, v, w \in V(G)$, such that $u \prec v \wedge v \prec w$ but $u \not\prec w$, or $w \prec u$. Since $C = 1$, the vertices $\langle u, v\rangle$, $\langle v, w\rangle$ and $\langle u, w\rangle$ must all be in the same component of $\mathcal{LVE}(G, \phi, \mathcal{L})$. There must exist a path P_1 from $\langle u, v\rangle$ to $\langle v, w\rangle$ and another path P_2 from $\langle v, w\rangle$ to $\langle u, w\rangle$. These two paths in $\mathcal{LVE}(G, \phi, \mathcal{L})$ induce two pairs of walks in G respectively; the first pair being W_1 from u to v and W_2 from v to w, and the second pair being W_3 from v to w and W_4 from w to u. Since the walks W_1 and W_2 (resp. W_3 and W_4) are induced by a single path P_1 (resp. P_2) in $\mathcal{LVE}(G, \phi, \mathcal{L})$, we can observe several facts regarding these pairs of walks:

- $\phi(W_1[i]) = \phi(W_2[i])$ (resp. $\phi(W_3[i]) = \phi(W_4[i])$) where $W[i]$ is the ith vertex along the walk W, *i.e.* they must progress through the levels in the same way;
- $W_1[i] \neq W_2[i]$ (resp. $W_3[i] \neq W_4[i]$);
- $W_1[i] \prec W_2[i]$ (resp. $W_3[i] \prec W_4[i]$).

So the walks in G, while not necessarily vertex or edge disjoint, must keep W_1 'to the left' of W_2 (resp. W_3 'to the left' of W_4). Also, W_1 and W_2 must

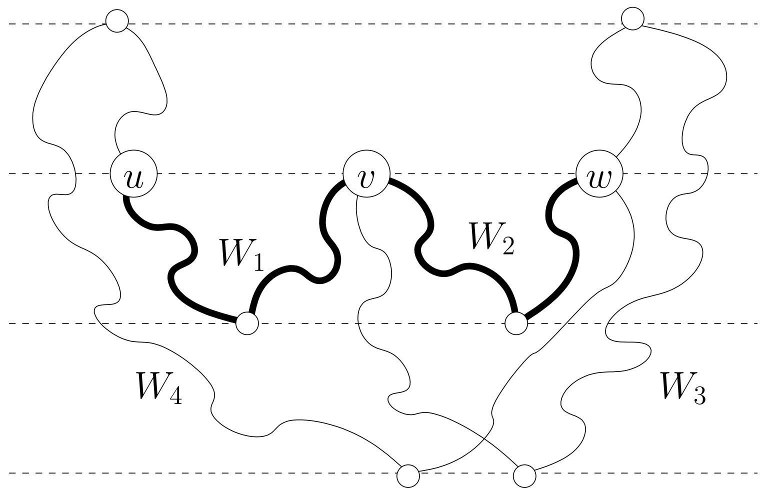


Figure 3.1: There must be at least one edge crossing involving the four walks, *e.g.* the bold pair do not cross so the lighter pair must.

finish where W_2 and W_3 start respectively, and W_2 and W_3 must finish were W_3 and W_1 start respectively. This is impossible and must result in at least one edge crossing (see Fig. 3.1) somewhere along the walks. But by Condition 1, we should have no edge crossings, and we are caught in a contradiction. Hence, Condition 2 must follow.

Case $C > 1$**:** Using the previous case, it is easy to show that, for each of the individual non-trivial components, Condition 1 follows.

We now show that Condition 2 follows. We can always reverse all relative orders within a single non-trivial component without violating Condition 1. This is not a problem in the previous case; Condition 2 will still automatically follow. However, if there is more than one non-trivial component, can we choose between the two alternatives for each so as to satisfy Condition 2? To prove sufficiency, we need only show that such a choice exists. We will leave the task of efficiently finding the choice until §3.2.2.

Instead of visiting each non-trivial component arbitrarily, we give priority to those that are *dependent* on previously visited non-trivial components. An unvisited component C_2 is dependent on a previously visited component C_1 if C_1 contains at least one and at most two vertices from the set $\{\langle u,v\rangle, \langle v,w\rangle, \langle u,w\rangle\} \subseteq V(\mathcal{LVE}(G,\phi,\mathcal{L}))$ where $u,v,w \in V(G)$ are three vertices on the same level of G and C_2 contains at most two of the missing vertices from the set. The relative order we have chosen for C_1 decides for us the relative order we must use for C_2 so that Condition 2 can be satisfied with the proviso that the following scenario cannot arise: Suppose there exists three vertices on the same level, $u, v, w \in V(G)$, and another three vertices on another level, $x, y, z \in V(G)$, such that the relative orders

- $u \prec v \wedge v \prec w$, and
- $z \prec y \wedge y \prec x$

have been chosen from previously visited components and $\langle u, w\rangle$ and $\langle z, x\rangle$ are about to be visited in the next component. Since the components containing $\langle u, v\rangle$, $\langle v, w\rangle$, $\langle x, y\rangle$ and $\langle y, z\rangle$ were all visited before the component containing $\langle u, w\rangle$ and $\langle z, x\rangle$, there must have been a dependence between them, involving some other vertex pair $p, q \in V(G)$ on some level of G.

We use a similar 'paths and walks argument' to that of the previous case. There must exist a path P_1 from $\langle u, v\rangle$ to $\langle v, w\rangle$, a path P_2 from $\langle z, y\rangle$ to $\langle y, x\rangle$, a path P_3 from $\langle u, w\rangle$ to $\langle p, q\rangle$, a path P_4 from $\langle z, x\rangle$ to $\langle p, q\rangle$, and finally a path P_5 from $\langle u, w\rangle$ to $\langle z, x\rangle$. These five paths in $\mathcal{LVE}(G, \phi, \mathcal{L})$ induce five pairs of walks in G respectively; the first pair being W_1 from u to v and W_2 from v to w, the second pair being W_3 from z to y and W_4 from y to x, the third pair being W_5 from u to p and W_6 from w to q, the fourth pair being W_7 from z to p and W_8 from x to q, and finally the fifth pair being W_9 from u to x and W_{10} from w to z. We can observe an analogous set of facts regarding these pairs of walks as in the previous case. Again, this situation is impossible and must result in at least one edge crossing (see Fig. 3.2) somewhere along the walks. But by Condition 1, we should have no edge crossings, and we are again caught in a contradiction. Hence, it must be possible to satisfy Condition 2.

$\square$

In other words, a graph G with leveling ϕ is level planar if and only if there exists some sequence of switches that removes all '–'-labeled edges from $\mathcal{LVE}(G, \phi, \mathcal{L})$. An equivalent characterization of balance (Harary and Kabell,

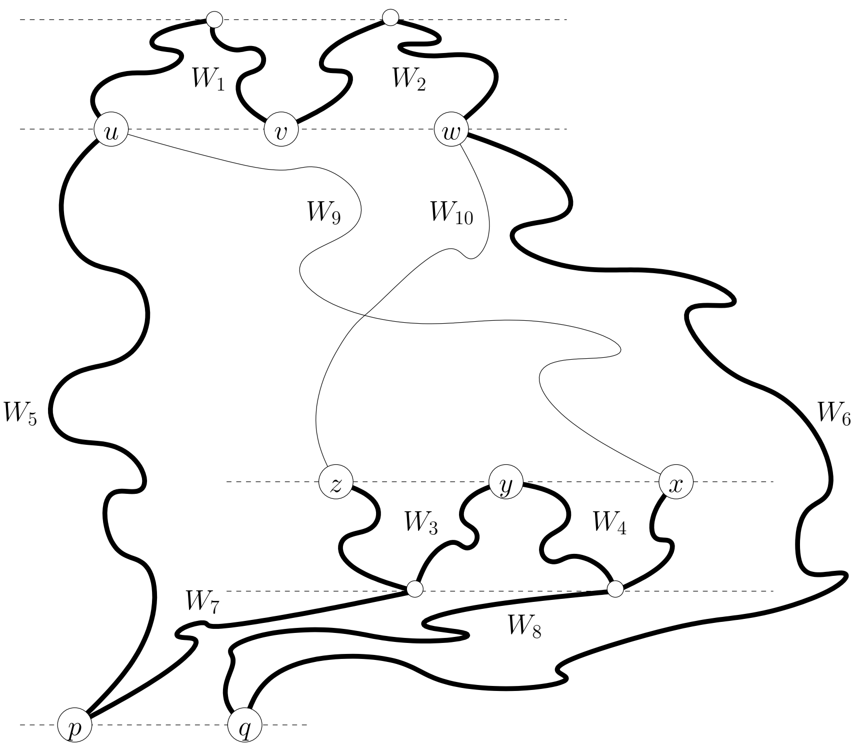


Figure 3.2: There must be at least one edge crossing involving the ten walks, *e.g.* the bold pairs do not cross so the lighter pair must.

1980; Loukakis, 2003) shows that a graph G with leveling ϕ is level planar if and only if G does not contain a cycle with an odd number of '–'-labeled edges. Level planarity can therefore be tested in quadratic running-time by first checking that the number of edges in $E(G)$ does not exceed the upper bound in Lemma 2.3.4 and then performing a simple depth-first search (DFS) traversal on $\mathcal{LVE}(G, \phi, \mathcal{L})$ (see Algorithm 1). We need to look for a cycle with an odd number of '–'-labeled edges. If one exists then G is not level planar, otherwise it is. However, in the case of level planarity, we need to solve the 3-*cycle problem* in order to produce a level planar embedding within the same asymptotic running-time.

3.2.1 The 3-Cycle Problem

Suppose three vertices $\langle u, v\rangle$, $\langle v, w\rangle$ and $\langle u, w\rangle$ in the labeled vertex-exchange graph representing the three vertices u, v and w in some leveled graph are not in the same component of the labeled vertex-exchange graph. If we perform

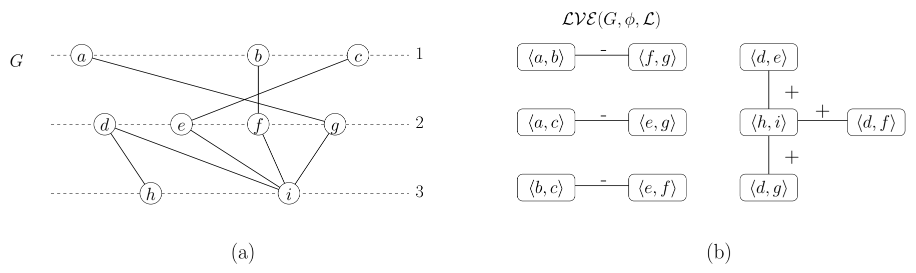


Figure 3.3: An instance of the 3-cycle problem.

switches to remove all the '–'-labeled edges using a DFS traversal then it may arise that $u \prec v \prec w \prec u$ where $\prec$ denotes the required order of the vertices along their respective level. Clearly, this is impossible.

Consider the graph G with leveling ϕ and its labeled vertex-exchange graph $\mathcal{LVE}(G,\phi,\mathcal{L})$ in Fig. 3.3. Note that $\langle e,f\rangle$, $\langle f,g\rangle$ and $\langle e,g\rangle$ are in different components of $\mathcal{LVE}(G,\phi,\mathcal{L})$. Suppose we begin the DFS traversal at $\langle e,f\rangle$, then visit $\langle b,c\rangle$ and perform `switch`($\langle b,c\rangle$) leaving $e \prec f$. Suppose we continue the DFS traversal at $\langle f,g\rangle$, then visit $\langle a,b\rangle$ and perform `switch`($\langle a,b\rangle$) leaving $f \prec g$. Now, suppose we continue the DFS traversal at $\langle a,c\rangle$, then visit $\langle e,g\rangle$ and perform `switch`($\langle e,g\rangle$) so that $g \prec e$. We have removed all '–'-labeled edges. However, $e \prec f \prec g \prec e$.

A book-keeping solution for every such triple of vertices in the leveled graph has been suggested (Healy and Kuusik, 2004). Every time we perform a switch that results in a vertex being constrained by another, we queue the constrained vertex and its respective component to be visited once we are finished with all the vertices in the current component. We are queuing the *dependent* components described in the proof of Theorem 3.2.1. Unfortunately there are $\mathcal{O}(n^3)$ such triples, leading to cubic running-time.

Randerath et al. (2001) have reduced the level planarity testing and embedding problems to satisfiability problems. They reduced the level planarity testing problem to a 2-SAT problem that is quadratic in the size of the leveled graph. However, if a level planar embedding is required, the solution must be 'enhanced' to avoid the 3-cycle problem (or, in their terminology, to satisfy the transitivity clauses). They show that such an enhancement is always possible but they do not show how to find it within the same asymptotic running-time.

3.2.2 The 3-Cycle Solution

We solve the 3-cycle problem using a combination of a DFS traversal and a level-by-level traversal. In this case the DFS traversal (Algorithm 1), given an initial leveled embedding $\mathcal{L}$, constructs a mapping π that tells us the required *relative* order of the vertices. For example, suppose $\langle u, v\rangle$ and $\langle w, x\rangle$ are in the same component of the labeled vertex-exchange graph. The algorithm may decide that $\pi(\langle u, v\rangle) = [v, u]$ and $\pi(\langle w, x\rangle) = [x, w]$. $[v, u]$ denotes an ordered pair of same-level vertices v and u. Note that $[u, v] \not\equiv [v, u]$. This means that $v \prec u \Leftrightarrow x \prec w$. It does not change the initial leveled embedding $\mathcal{L}$. If the DFS traversal returns `false` then it has found a cycle with an odd number of '–'-labeled edges (Lines **23** and **27**) and the leveled graph is not level planar. If the DFS traversal returns `true` then the leveled graph is level planar and we can proceed to the level-by-level traversal to produce a level planar embedding.

Before describing the level-by-level traversal, we need to examine the nature of a switch when performed on $\mathcal{LVE}(G, \phi, \mathcal{L})$. In particular, how closely does `switch`$(\langle u, v\rangle)$ correspond to exchanging the position of two vertices $u, v \in \phi^{-1}(j)$ in the leveled embedding $\mathcal{L}$ of G? If u and v do not have any other vertices in between them on level j then the correspondence is exact. If we exchange the position of u and v in $\mathcal{L}$ then the only edges in $\mathcal{LVE}(G, \phi, \mathcal{L})$ that change label are those that represent pairs of independent edges that have u and v as two of their end points. These are precisely the edges that are incident with $\langle u, v\rangle$ (see Fig. 3.4).

The situation is not as favorable when there are vertices in between u and v on level j. As can be seen from Fig. 3.5 when exchanging d and g, the correspondence no longer holds. The edge labelings surrounded by stars should also have changed. In order to keep the labeled vertex-exchange graph consistent with the leveled graph, we need to perform a sequence of switches of the previous form. For the leveled graph in Fig. 3.5, we can perform switches on

```
Input: $\mathcal{VE}(G,\phi), \mathcal{L}, \pi$ (passed by reference)
Output: $\{$true, false$\}$
1  $visited \leftarrow \{\}$;  /* a mapping from previously visited components
   to the relative order of the first vertex visited in each */
2  Initialize a stack $S$;
3  foreach $C \in$ connectedComponents$(\mathcal{VE}(G,\phi))$ do
4      Choose some vertex $\langle u,v\rangle$ in $C$;
5      if $u \prec v$ then
6          $\pi(\langle u,v\rangle) \leftarrow [u,v]$;
7      else $\pi(\langle u,v\rangle) \leftarrow [v,u]$;
8      push$(S, \langle u,v\rangle,$ false$)$;
9  while $S$ not empty do
10     $\langle u,v\rangle, value \leftarrow$ pop$(S)$;
11     $visited(\langle u,v\rangle) \leftarrow$ true;
12     foreach $\langle w,x\rangle \in$ neighbors$(\langle u,v\rangle)$ do
13         if $\lambda(\{\langle u,v\rangle,\langle w,x\rangle\}) =$ '+' then
14             $p \leftarrow [w,x]$, $q \leftarrow [x,w]$;
15             if $visited(\langle w,x\rangle) =$ false then
16                 push$(S, \langle w,x\rangle, value)$;
17         else
18             $p \leftarrow [x,w]$, $q \leftarrow [w,x]$;
19             if $visited(\langle w,x\rangle) =$ false then
20                 push$(S, \langle w,x\rangle, \neg value)$;
21         if $(w \prec x \wedge \neg value) \vee (x \prec w \wedge value)$ then
22             if $visited(\langle w,x\rangle) =$ true $\wedge\ \pi(\langle w,x\rangle) \neq p$ then
23                 return false;
24             $\pi(\langle w,x\rangle) \leftarrow p$;
25         else
26             if $visited(\langle w,x\rangle) =$ true $\wedge\ \pi(\langle w,x\rangle) \neq q$ then
27                 return false;
28             $\pi(\langle w,x\rangle) \leftarrow q$;
29 return true;
```

Algorithm 1: dfsTraversal

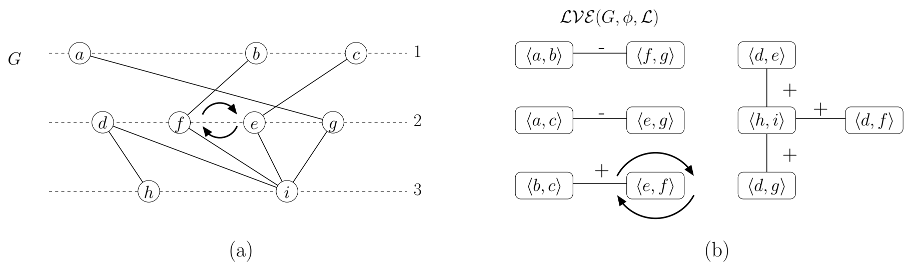


Figure 3.4: `switch`($\langle u, v\rangle$) when u and v do not have any other vertices in between them on level j. Note the change in $\mathcal{LVE}(G, \phi, \mathcal{L}$ from Fig. 3.3

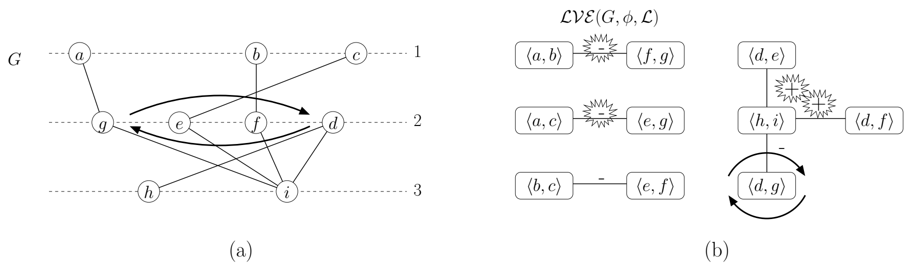


Figure 3.5: `switch`($\langle u, v\rangle$) when u and v do have other vertices in between them on level j. The inconsistent edge labelings are marked with stars.

$\langle d, e\rangle$, $\langle d, f\rangle$, $\langle d, g\rangle$, $\langle f, g\rangle$, and $\langle e, g\rangle$ (in that order). Each switch is performed on a vertex whose corresponding vertices in the leveled graph do not have any other vertices in between them on the same level at the time of the switch. This guarantees the consistency of the labeled vertex-exchange graph throughout. The overall effect of these switches is to exchange the positions of $d, g \in \phi^{-1}(2)$. More generally, to exchange the position of two vertices $u, v \in \phi^{-1}(j)$ in a leveled embedding $\mathcal{L}$ of G where $u_1, u_2, u_3, \ldots, u_{p-2}, u_{p-1}, u_p$ are the vertices in between u and v, we perform switches on:

- $\langle u, u_1\rangle, \langle u, u_2\rangle, \langle u, u_3\rangle, \ldots, \langle u, u_{p-2}\rangle, \langle u, u_{p-1}\rangle, \langle u, u_p\rangle$;
- $\langle u, v\rangle$;
- $\langle u_p, v\rangle, \langle u_{p-1}, v\rangle, \langle u_{p-2}, v\rangle, \ldots, \langle u_3, v\rangle, \langle u_2, v\rangle, \langle u_1, v\rangle$

(in that order). The same leveled graph as in Fig. 3.5 is shown again in Fig. 3.6 along with the correctly labeled vertex-exchange graph after such a sequence of switches.

These ‘switch sequences’ raise two more issues if used by the putative level-by-level traversal algorithm: Can we still keep the running-time of the traversal

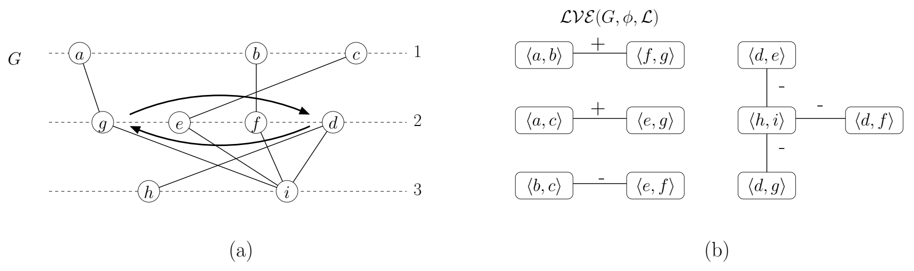


Figure 3.6: The same leveled graph as in Fig. 3.5 with the correctly labeled vertex-exchange graph.

linear in the size of the labeled vertex-exchange graph? Can we traverse the labeled vertex-exchange graph in such a way that performing a switch in one part will not adversely affect some other part, *i.e.* by introducing '–'-labels along edges we have already traversed?

We can answer the first question affirmatively by determining the label of an edge dynamically instead of pre-computing a labeled vertex-exchange graph and updating the labels every time we change $\mathcal{L}$. To dynamically determine the label of an edge, we simply find its two corresponding edges in the leveled graph and check if they cross or not (using Observation 2.3.1). When referring to the labeled vertex-exchange graph in the remainder of this section, it is assumed that we only store the vertex-exchange graph (without labels) and determine the actual labels as required. The '+'- and '–'-labels in all the figures with labeled vertex-exchange graphs are for illustrative purposes only; they are not stored with the data structure but computed on the fly. The second question can also be answered affirmatively using the level-by-level traversal.

The level-by-level traversal (Algorithm 2) makes $\mathcal{L}$ level planar one level at a time by deciding on the *absolute* order of the vertices in the final leveled embedding. The vertices of the labeled vertex-exchange graph are grouped by the level of the vertices in the leveled graph they represent. We visit the vertices in each group $1, \ldots, k$. Within each group the vertices are visited in descending order according to the distance between the vertices they represent in the leveled embedding $\mathcal{L}$ of the leveled graph. This traversal proceeds left-to-right and top-to-bottom along the dotted lines in Fig. 3.7(b). It is controlled by $\texttt{rows}(\mathcal{L})$ (the number of dotted lines), $\texttt{cols}(\mathcal{L}, i)$ (the number of vertices on

```
Input: $\mathcal{VE}(G, \phi), \mathcal{L}$ (passed by reference), $\pi$
1  $visited \leftarrow \{\}$;
2  for $i$ from 1 to $\mathtt{rows}(\mathcal{L})$ do
3      for $j$ from 1 to $\mathtt{cols}(\mathcal{L}, i)$ do
4          $\langle u, v\rangle \leftarrow \mathtt{vertexAt}(\mathcal{L}, i, j)$;
5          $plusCnt \leftarrow 0$;
6          $minusCnt \leftarrow 0$;
7          foreach $w \in \mathtt{neighbors}(\langle u, v\rangle)$ do
8              if $\mathtt{row}(w) < i \vee (\mathtt{row}(w) = i \wedge \mathtt{col}(w) \leq j)$ then
9                  if $\lambda(\{\langle u, v\rangle, w\}) =$ '+' then
10                     $plusCnt \leftarrow plusCnt + 1$;
11                 else $minusCnt \leftarrow minusCnt + 1$;
12         $C \leftarrow \mathtt{connectedComponent}(\mathcal{VE}(G, \phi), \langle u, v\rangle)$;
13         if $C \notin \mathtt{domain}(visited)$ then
14             if $u \prec v$ then
15                 $visited(C) \leftarrow [u, v]$;
16             else $visited(C) \leftarrow [v, u]$;
17         else if
           $plusCnt + minusCnt = 0 \wedge \neg\mathtt{match}(\mathcal{L}, \pi, visited, C, \langle u, v\rangle)$ then
18             Exchange $u$ and $v$ in $\mathcal{L}$, i.e. $u \prec v$ becomes $v \prec u$ and $v \prec u$
               becomes $u \prec v$;
19         if $minusCnt > 0$ then
20             Exchange $u$ and $v$ in $\mathcal{L}$, i.e. $u \prec v$ becomes $v \prec u$ and $v \prec u$
               becomes $u \prec v$;
```

Algorithm 2: `levelByLevelTraversal`

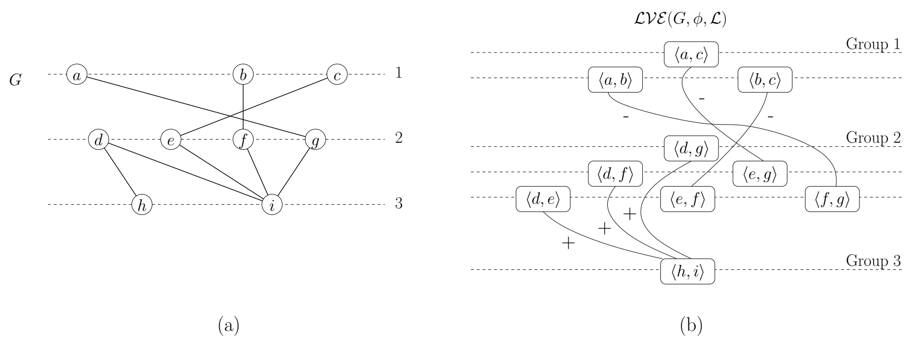


Figure 3.7: (a) The initial leveled embedding $\mathcal{L}$ of G and (b) the grouping of the vertices of $\mathcal{LVE}(G, \phi, \mathcal{L})$).

the ith dotted line) and `vertexAt`$(\mathcal{L}, i, j)$ (the vertex at the jth position on the ith dotted line). Note that the position of some vertices may change after each update of $\mathcal{L}$. However, the traversal proceeds in this direction irrespectively.

On visiting a vertex $\langle u, v\rangle$, we count the number of neighbors it has in the previous group that are joined by '+' (`plusCnt`) and '–'-labeled edges (`minusCnt`). As previously noted, the labels of the edges of the labeled vertex-exchange graph are computed dynamically. Line **9** of Algorithm 2 will involve a function call to determine its label. There are three cases:

1. **Lines 13 to 16:** $\langle u, v\rangle$ belongs to a hitherto unvisited component C of the labeled vertex-exchange graph: We mark C as visited and record whether $u \prec v$ or $v \prec u$.

2. **Lines 17 to 18:** $\langle u, v\rangle$ belongs to a previously visited component C of the labeled vertex-exchange graph and `plusCnt` + `minusCnt` $= 0$: We use π to decide whether or not we need to exchange u and v in $\mathcal{L}$. Let $\langle w, x\rangle$ be the first vertex visited in C and *visited* be a mapping from previously visited components to the relative order of the first vertex visited in each. Our test `match`$(\mathcal{L}, \pi, \mathit{visited}, C, \langle u, v\rangle)$ returns `true` if any of the following are true:

   - $\pi(\langle w, x\rangle) = \mathit{visited}(C)$ and $u \prec v$ and $\pi(\langle u, v\rangle) = [u, v]$;
   - $\pi(\langle w, x\rangle) = \mathit{visited}(C)$ and $v \prec u$ and $\pi(\langle u, v\rangle) = [v, u]$;
   - $\pi(\langle w, x\rangle) \neq \mathit{visited}(C)$ and $u \prec v$ and $\pi(\langle u, v\rangle) = [v, u]$;

- $\pi(\langle w,x\rangle) \neq \mathit{visited}(C)$ and $v \prec u$ and $\pi(\langle u,v\rangle) = [u,v]$;

and `false` otherwise. `match` tells us if the vertices u and v are already in the correct order. It so happens that $\pi(\langle w,x\rangle) = \mathit{visited}(C)$ always since we never exchange u and v in the previous case, but we include it here for completeness. This test is negated in Line **17** because we only want to exchange u and v in $\mathcal{L}$ if the present order is incorrect.

3. **Lines 19 to 20:** $\langle u,v\rangle$ belongs to a previously visited component C of the labeled vertex-exchange graph and `minusCnt` > 0. It cannot arise that `minusCnt` > 0 and `plusCnt` > 0 since this would mean the presence of a cycle with an odd number of '–'-labeled edges. Hence `plusCnt` $= 0$. We exchange u and v in $\mathcal{L}$.

We now illustrate this process with a worked example before returning to its rationale. Consider the leveled graph and its labeled vertex-exchange graph in Fig. 3.7. We first perform `dfsTraversal`$(\mathcal{VE}(G,\phi),\mathcal{L},\pi)$ to obtain the following relative orders for the four components:

- $\pi(\langle b,c\rangle) = [c,b]$ and $\pi(\langle e,f\rangle) = [e,f]$;
- $\pi(\langle a,b\rangle) = [b,a]$ and $\pi(\langle f,g\rangle) = [f,g]$;
- $\pi(\langle e,g\rangle) = [g,e]$ and $\pi(\langle a,c\rangle) = [a,c]$;
- $\pi(\langle d,e\rangle) = [d,e]$, $\pi(\langle h,i\rangle) = [h,i]$, $\pi(\langle d,f\rangle) = [d,f]$, and $\pi(\langle d,g\rangle) = [d,g]$.

We then perform `levelByLevelTraversal`$(\mathcal{VE}(G,\phi),\mathcal{L})$. The first four visited vertices are $\langle a,c\rangle$, $\langle a,b\rangle$, $\langle b,c\rangle$ and $\langle d,g\rangle$. These all match Case 1 above and so no changes are made to $\mathcal{L}$ (see Fig. 3.8). Next we visit $\langle d,f\rangle$. This vertex matches Case 2 since it belongs to a previously visited component and it has no neighbors in the previous group. Our test `match`$(\mathcal{L},\pi,\mathit{visited},C,\langle u,v\rangle)$ returns `true` meaning d and f are already in the correct order and so, again, no changes are made to $\mathcal{L}$ (see Fig. 3.9). Next we visit $\langle e,g\rangle$. This vertex matches

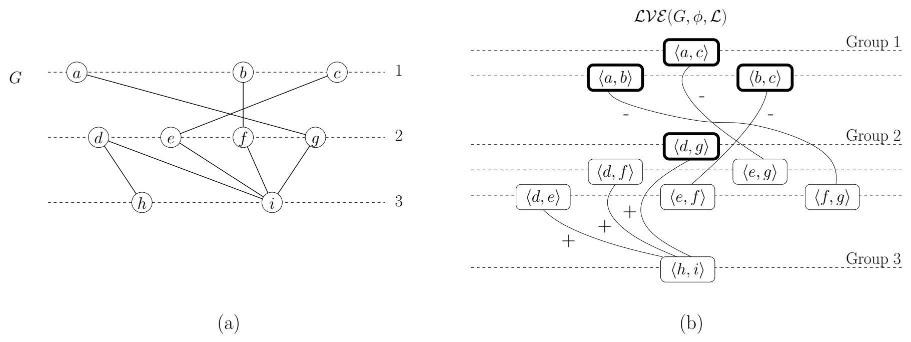


Figure 3.8: (a) The leveled embedding $\mathcal{L}$ remains unchanged after (b) visiting the first four vertices of $\mathcal{LVE}(G,\phi,\mathcal{L})$) (all Case 1).

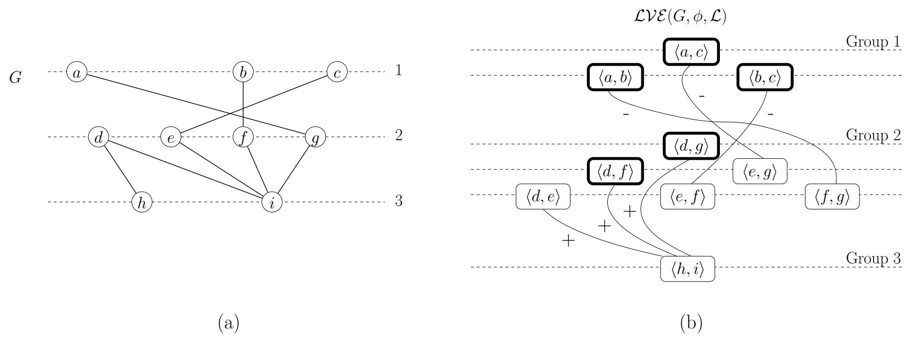


Figure 3.9: (a) The leveled embedding $\mathcal{L}$ remains unchanged after (b) visiting $\langle d,f\rangle$ (Case 2).

Case 3 since it belongs to a previously visited component and `minusCnt` > 0. We exchange e and g in $\mathcal{L}$ (see Fig. 3.10). This changes three of the edge labelings from '+' to '–' and the positions of the vertices $\langle d,e\rangle$, $\langle d,g\rangle$, $\langle f,g\rangle$ and $\langle e,f\rangle$ in Group 2. The last four visited vertices are $\langle d,g\rangle$, $\langle f,g\rangle$, $\langle e,f\rangle$ and $\langle h,i\rangle$ (see Fig. 3.11). These all match Case 1 above and so no further changes are made to $\mathcal{L}$. It is interesting to note that, in the course of the traversal, $\langle d,g\rangle$ was visited twice whereas $\langle d,e\rangle$ was never visited. This does not adversely affect the asymptotic running-time since the overall number of visited positions will always be $\mathcal{O}(|V(\mathcal{VE}(G,\phi))|)$.

3.2.3 Proof of Correctness

During each iteration of the inner `for` loop (Lines **3** to **20**) of Algorithm 2, the vertex at position i,j (`vertexAt`$(\mathcal{L},i,j)$) is being visited. This is the one and only time this position is visited. Lines **18** and **20** within this iteration are the

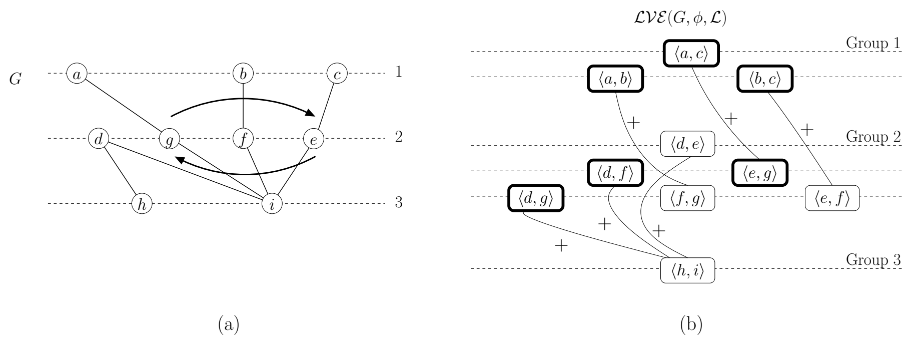


Figure 3.10: (a) We exchange e and g in $\mathcal{L}$ and (b) re-locate some of the vertices in Group 2 (Case 3).

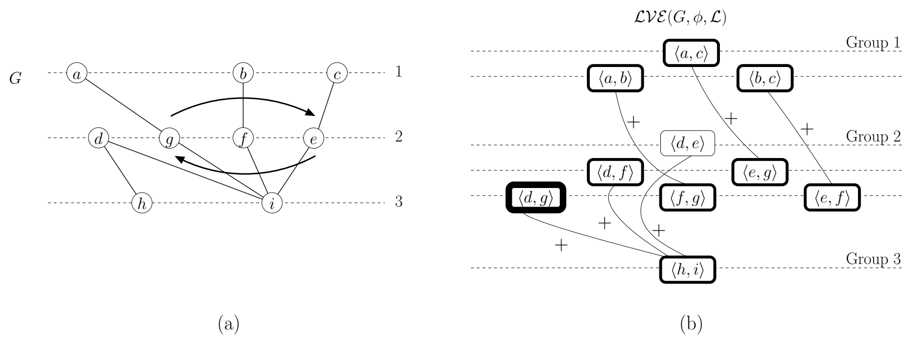


Figure 3.11: (a) The leveled embedding $\mathcal{L}$ remains unchanged after (b) visiting the last four vertices of $\mathcal{LVE}(G, \phi, \mathcal{L})$) (all Case 1).

only places in which $\mathcal{L}$ and $\mathcal{LVE}(G, \phi, \mathcal{L})$ may change during this iteration. Moreover, we can show the following:

Theorem 3.2.2. *At the end of each iteration of the inner* `for` *loop (Lines **3** to **20**) of Algorithm 2, all edges of $\mathcal{LVE}(G, \phi, \mathcal{L})$ with end points in positions i_1, j_1 and i_2, j_2 where $i_1, i_2 \leq i$, $j_1 \leq j$ if $i_1 = i$, $j_2 \leq j$ if $i_2 = i$ and i, j is the position being visited by the present iteration, are '+'-labeled.*

Proof: The proof is inductive in the position i, j being visited by the inner `for` loop.

Position $1, 1$**:** This is trivially true since there are no edges of $\mathcal{LVE}(G, \phi, \mathcal{L})$ with end points in positions preceding $1, 1$.

Position i, j**:** We assume the theorem is true for all positions i', j' where $i' < i$ or, $i' = i$ and $j' < j$, where the vertex $\langle u, v \rangle$ at position i, j is presently being visited.

If Case 1 applies (Lines **13** to **16** of Algorithm 2) then no change is made to $\mathcal{L}$ or $\mathcal{LVE}(G, \phi, \mathcal{L})$ and the theorem holds.

If Case 2 applies (Lines **17** to **18** of Algorithm 2) then a change is made to $\mathcal{L}$ and $\mathcal{LVE}(G, \phi, \mathcal{L})$ at Line **18**. Since `plusCnt` + `minusCnt` $= 0$, $\langle u, v \rangle$ does not have any neighbors in the previous group. Any edges incident with u or v from the previous level cannot be independent (see Fig. 3.12(a)). If we assume, w.l.o.g, that $u \prec v$ before any change is made, then we can enumerate all possible classes of positions for edges in G whose corresponding edges in $\mathcal{LVE}(G, \phi, \mathcal{L})$ may change label as a result of exchanging u and v in $\mathcal{L}$. There are (ignoring symmetry) four such classes of positions (illustrated by the four lighter edges in Fig. 3.12(a)): (i) the edges with one end point preceding u and the other preceding all vertices adjacent to both u and v, (ii) the edges with one end point between u and v and the other preceding all vertices adjacent to both u and v, (iii) the edges with one end point between u and v and the other succeeding all vertices adjacent to both u and v, and finally (iv) the edges with one end point preceding u and the other succeeding all vertices adjacent to both u and v (the dotted line in Fig. 3.12(a)). It turns out that

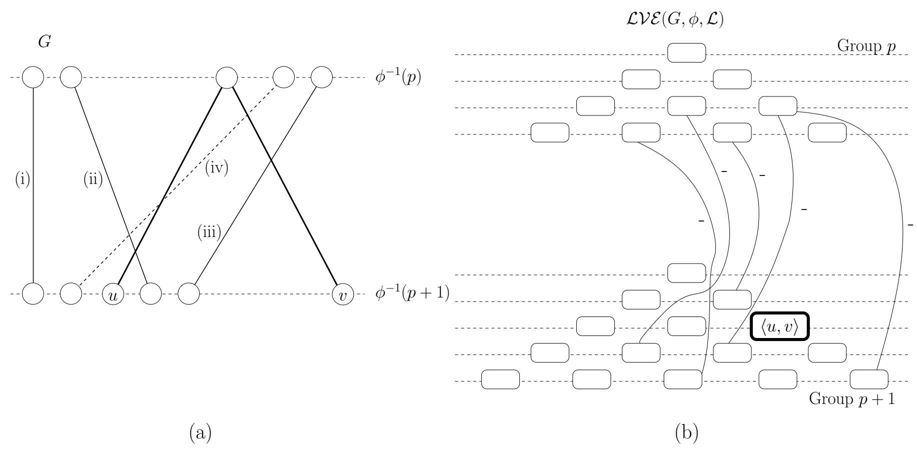


Figure 3.12: (a) There are (ignoring symmetry) four possible classes of positions for the edges in G whose corresponding edges in $\mathcal{LVE}(G,\phi,\mathcal{L})$ may change label due to Line **18**. (b) $\mathcal{LVE}(G,\phi,\mathcal{L})$ with only the '–'-labeled edges shown.

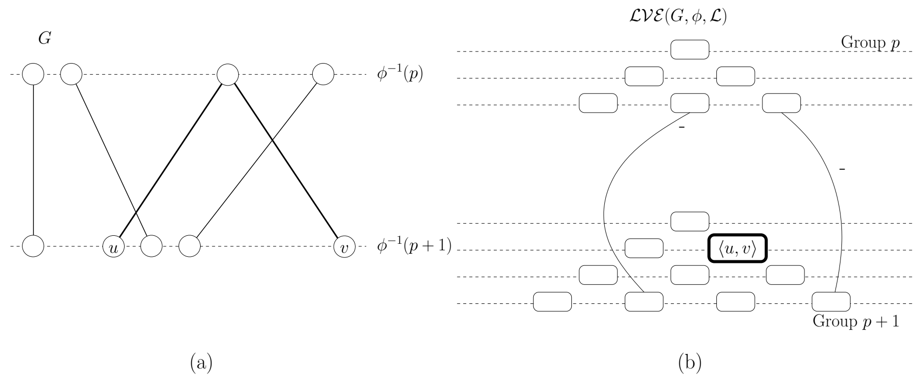


Figure 3.13: A simplified version of Fig. 3.12 with the dotted edge that violates the inductive assumption removed.

this last possibility cannot occur, since it violates our inductive assumption: it leads to a '–'-labeled edge in $\mathcal{LVE}(G,\phi,\mathcal{L})$ with an end point at position i', j' where $i' < i$ and $j' < j$ (see Fig. 3.12(b)). Removing this possibility simplifies matters slightly (see Fig. 3.13(a) and (b)). It is easy to see that exchanging u and v in $\mathcal{L}$ will not introduce any new '–'-labeled edges with end point at positions i_1, j_1 and i_2, j_2 where $i_1, i_2 \leq i$ and $j_1, j_2 \leq j$. In fact, $\mathcal{LVE}(G,\phi,\mathcal{L})$ will remain exactly as in Fig. 3.13(b).

If Case 3 applies (Lines **19** to **20** of Algorithm 2) then a change is made to $\mathcal{L}$ and $\mathcal{LVE}(G,\phi,\mathcal{L})$ at Line **20**. The argument is akin to the previous case, except there are many more possible classes of positions for edges in G to consider. Since `minusCnt` > 0, $\langle u,v\rangle$ does have at least one neighbor in the previous group joined by only '–'-labeled edges. If we assume, w.l.o.g, that $u \prec v$ before any change is made, then we can enumerate all possible classes of positions for edges in G whose corresponding edges in $\mathcal{LVE}(G, \phi, \mathcal{L})$ may change label as a result of exchanging u and v in $\mathcal{L}$.

There are (ignoring symmetry) ten such classes of positions (illustrated by the ten lighter edges in Fig. 3.14): (i) the edges with one end point preceding u and the other preceding all vertices adjacent to either u or v, (ii) the edges with one end point between u and v and the other preceding all vertices adjacent to either u or v, (iii) the edges with one end point between u and v and the other between any vertex adjacent to u and any vertex adjacent to v, (iv) the edges with one end point between u and v and the other succeeding all vertices adjacent to either u or v, (v) the edges with u as one end point and the other preceding all vertices adjacent to either u or v, (vi) the edges with u as one end point and the other adjacent to v, (vii) the edges with u as one end point and the other between any vertex adjacent to u and any vertex adjacent to v, (viii) the edges with u as one end point and the other succeeding all vertices adjacent to either u or v, (ix) the edges with one end point preceding u and the other between any vertex adjacent to u and any vertex adjacent to v, and finally (x) the edges with one end point preceding u and the other succeeding all vertices adjacent to either u or v. It turns out that last two of these possibilities (the dotted lines in Fig. 3.14) cannot occur, since they violate our inductive assumption.

Removing these edges we get the leveled graph in Fig. 3.16 and a simplification of the labeled vertex-exchange graph from the one in Fig. 3.15 to the one in Fig. 3.17.

Exchanging u and v in $\mathcal{L}$ will introduce just one new '–'-labeled (class of) edge with end points that violate the inductive hypothesis (see Fig. 3.18 and Fig. 3.19). This edge is incident with $\langle u, v\rangle$ and must have been '+'-labeled before the switch. However, according to Case 3 such an edge cannot exist since if `minusCnt` > 0 then `plusCnt` $= 0$.

□

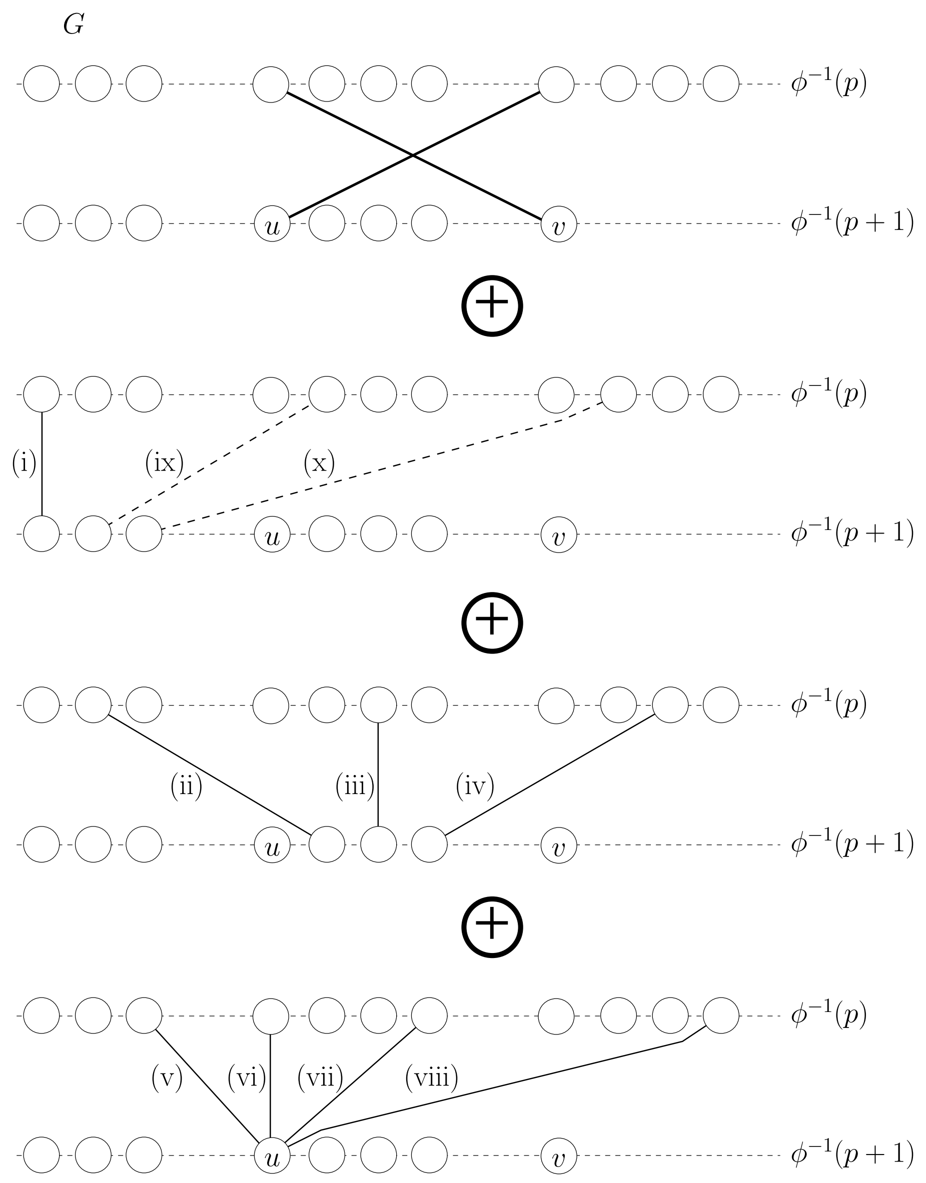


Figure 3.14: There are (ignoring symmetry) ten possible classes of positions for the edges in G whose corresponding edges in $\mathcal{LVE}(G, \phi, \mathcal{L})$ may change label due to Line **20**. In this, and the following figures, the circled plus indicates that the individual drawings must be superimposed on one another (like overlays) to get the entire drawing.

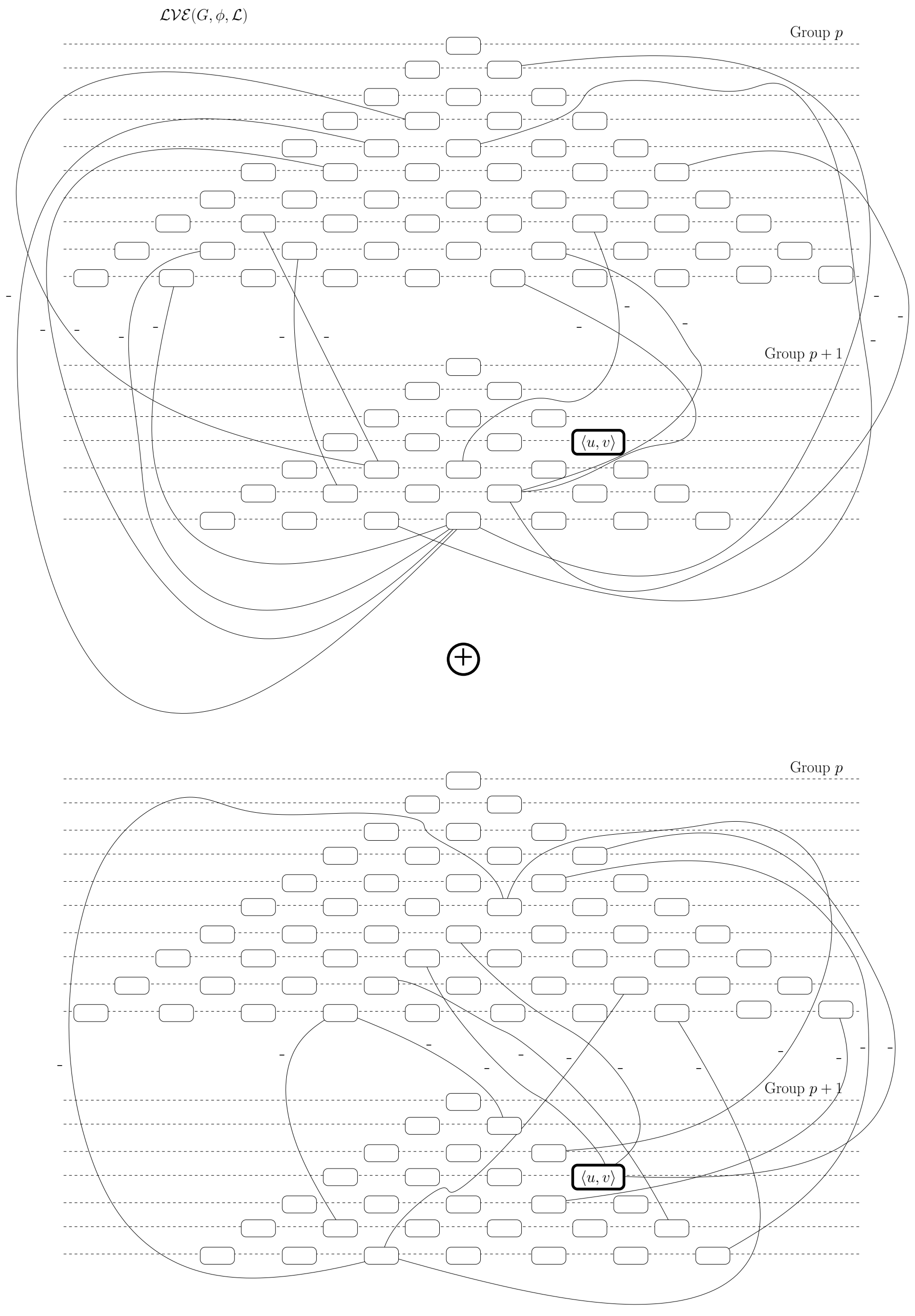


Figure 3.15: $\mathcal{LVE}(G,\phi,\mathcal{L})$ for the leveled graph in Fig. 3.14 with only the '–'-labeled edges shown.

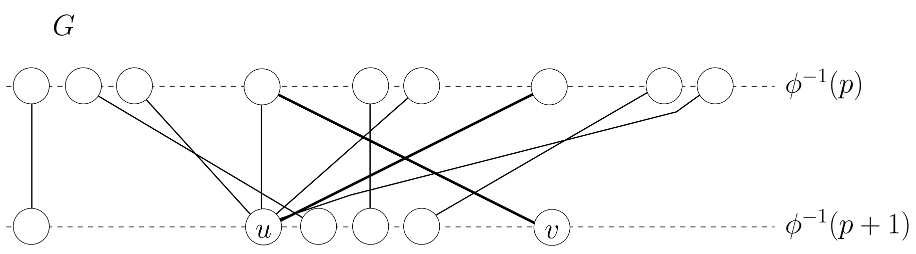


Figure 3.16: A simplified version of Fig. 3.14 with the dotted edges that violate the inductive assumption removed.

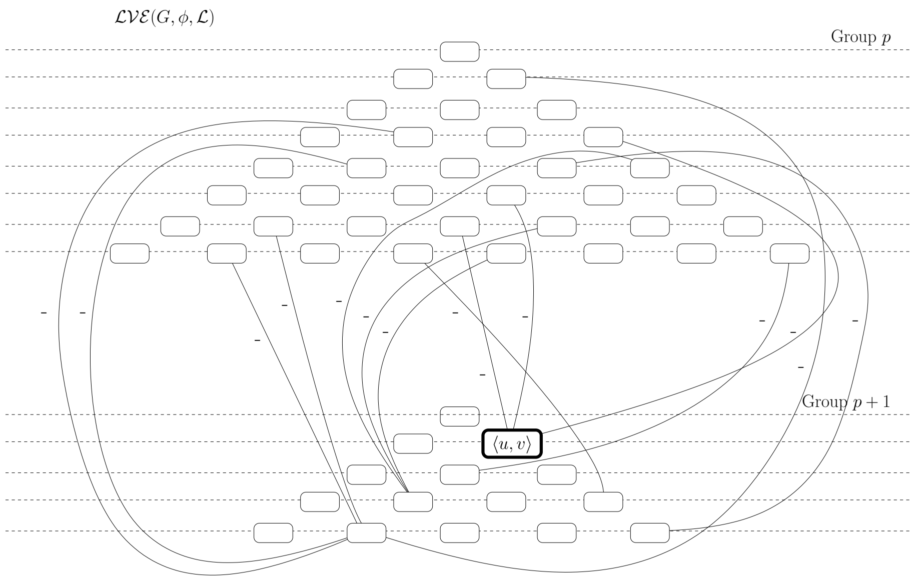


Figure 3.17: A simplified $\mathcal{LVE}(G,\phi,\mathcal{L})$ for the leveled graph in Fig.3.16.

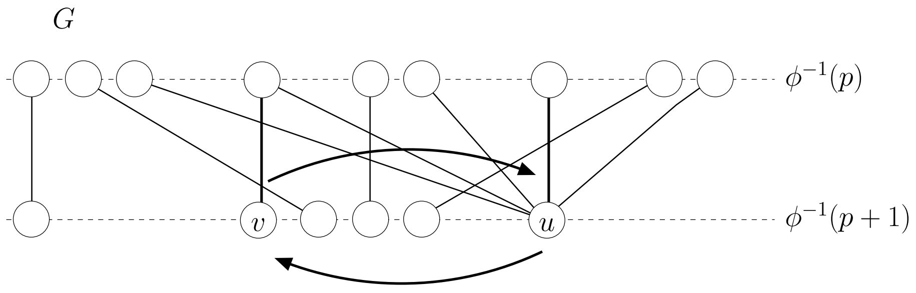


Figure 3.18: Exchanging u and v in $\mathcal{L}$.

This proves our earlier claim that Algorithm 2 makes $\mathcal{L}$ level planar one level (or one group of $\mathcal{LVE}(G,\phi,\mathcal{L})$ vertices) at a time and hence, the correctness of the entire algorithm must follow.

3.3 Extensions

In the following sections, we show how our approach to level planarity testing and embedding can be extended to handle *cyclic leveled graphs* (§3.3.1), *embedding constraints* (§3.3.2), and tackle the k-level crossing minimization problem (§3.3.3).

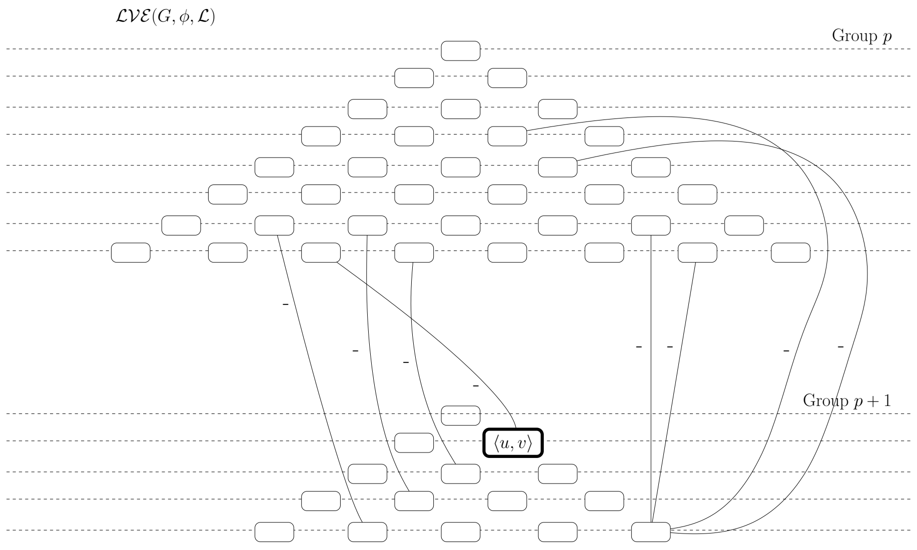


Figure 3.19: $\mathcal{LVE}(G, \phi, \mathcal{L})$ for the leveled graph in Fig. 3.18. Note that Theorem 3.2.2 holds.

3.3.1 Cyclic Level Planarity Testing and Embedding

Cyclic leveled graphs (or *recurrent hierarchies* (Sugiyama et al., 1981)) are leveled graphs where the first level is the successor of the last level (see Fig. 3.20). In particular, a *cyclic leveling* is *proper* if $\forall \{u, v\} \in E(G) : |\phi(u) - \phi(v)| \equiv 1 \mod (k-2)$ where k is the number of levels in ϕ. Testing a cyclic leveled graph for *cyclic level planarity* and producing a *cyclic level planar embedding* if the test succeeds was considered by Bachmaier et al. (2008). Their method relies heavily on the linear time level planarity testing and embedding algorithms of Jünger and Leipert (2002) (§3.1) and is, perforce, quite involved. However, it is a simple matter to extend our testing and embedding algorithms to handle cyclic leveled graphs: we simply note that a proper cyclic leveling partitions the edge set $E(G) = E_1 \cup E_2 \cup \ldots \cup E_{k-1} \cup E_k$ and treat the edges in E_k (from level k to level 1) like any other when computing the (labeled) vertex-exchange graph.

3.3.2 Embedding Constraints

We now extend our testing and embedding algorithms to support the expression and satisfaction of constraints (Tamassia, 1998). In particular, we consider

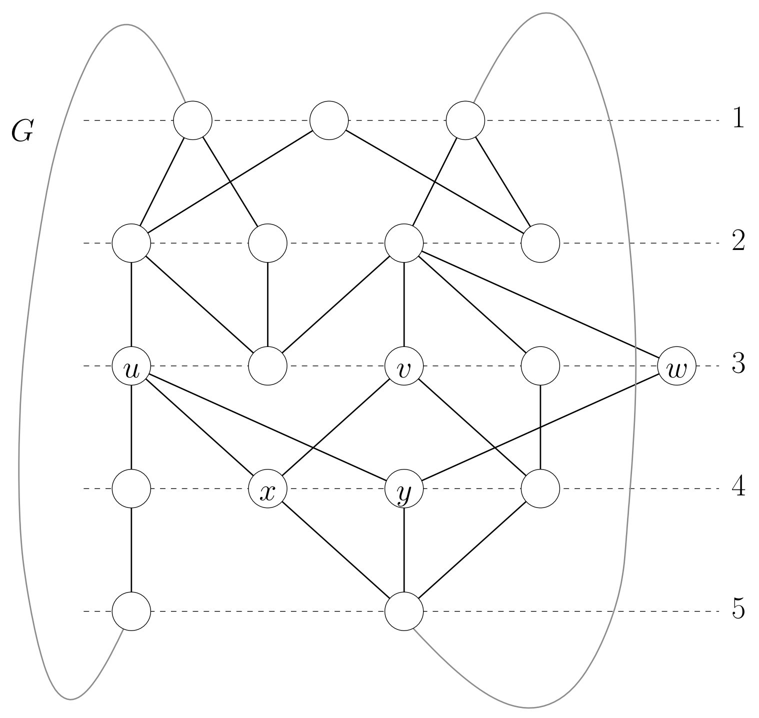


Figure 3.20: A cyclic k-leveled drawing of a graph G with cyclic leveling ϕ. The two lighter edges are 'wrapped' around the back of the drawing.

embedding constraints, whereby we restrict the order of incident edges around certain vertices. In other words, we wish to test a leveled graph for level planarity and produce a level planar embedding if the test succeeds subject to having the order of incident edges around certain vertices constrained.

We use *constraint trees* (Gutwenger et al., 2007) to express embedding constraints. Given a graph G with leveling ϕ, a constraint tree $T(v)$ (see Fig. 3.21(b)) with respect to some vertex $v \in V(G)$ is an ordered tree rooted at v where inner vertices (except the root) impose constraints on their children and the leaves of $T(v)$ are v's incident edges between the levels $\phi(v)$ and $\phi(v)-1$ or $\phi(v)$ and $\phi(v)+1$, depending on whether the incident edges we wish to constrain join v to vertices on the previous or successive level.

There are three types of inner vertex:

1. *grouping constraint vertices* or *gc-vertices* allow their children to be permuted arbitrarily in $T(v)$;

2. *mirroring constraint vertices* or *mc-vertices* allow their children to be

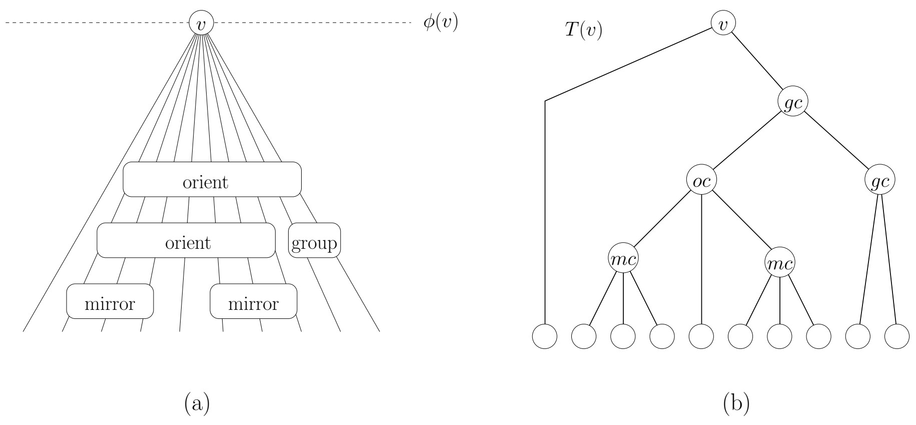


Figure 3.21: (a) We would like to restrict the order of v's incident edges. (b) The constraint tree $T(v)$ expresses these restrictions.

reversed in $T(v)$;

3. *orientation constraint vertices* or *oc-vertices* fix the order of their children in $T(v)$.

Figure 3.21(a) shows the desired restrictions we would like to place on the order of the incident edges of some vertex v. $T(v)$ in Fig. 3.21(b) expresses these constraints by admitting only certain orderings of its leaves. An ordering is admissible if we can produce it as some left-to-right permutation of $T(v)$'s leaves by permuting the children of the gc-vertices arbitrarily and reversing the children of the mc-vertices only. The order of the children of the oc-vertices is fixed.

For every constraint tree $T(v)$, we expand the leveled graph G by replacing the subgraph induced by v and its neighbors in the previous (or successive) level with $T(v)$ to get G'. We also expand ϕ so that the inner vertices belong to 'sub-levels' between $\phi(v)$ and $\phi(v) - 1$ (or $\phi(v)$ and $\phi(v) + 1$) to get ϕ' (see Fig. 3.22).

We compute $\mathcal{LVE}(G', \phi', \mathcal{L}')$ with respect to some initial leveled embedding $\mathcal{L}'$. The edges of the constraint trees are not allowed to cross in this initial leveled embedding. We treat the inner vertices and sub-levels as regular vertices and levels respectively but with the following additions:

- Every subgraph induced by vertices in $V(\mathcal{LVE}(G', \phi', \mathcal{L}'))$ whose corre-

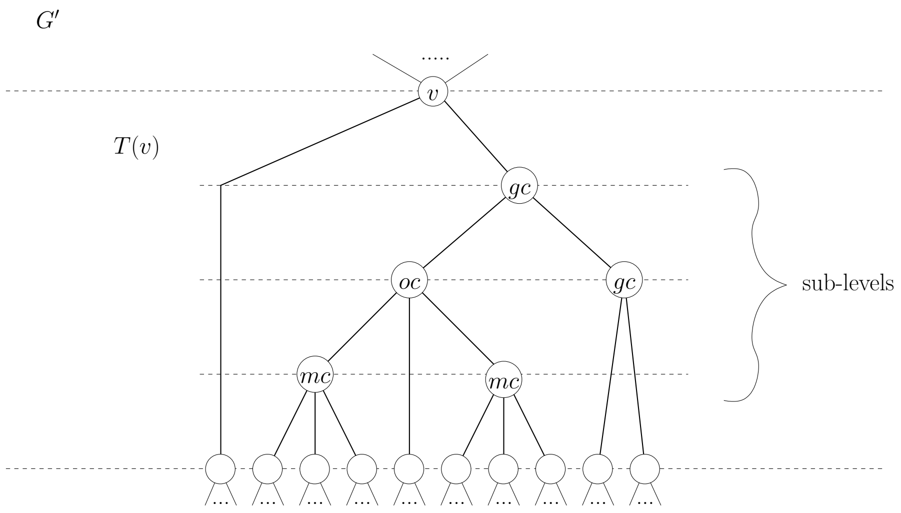


Figure 3.22: We expand G and ϕ by placing $T(v)$ directly between the two levels containing the edges we are trying to constrain.

sponding vertices in $V(G')$ are children of the same mc-vertex is made biconnected using '+'-labeled edges, *e.g.* by 'stringing' a cycle through the vertices. These extra edges restrict the orders of the children of the mc-vertices to being reversed altogether or not at all.

- A '+'-labeled loop is added to every vertex in $V(\mathcal{LVE}(G', \phi', \mathcal{L}'))$ whose corresponding vertices in $V(G')$ are children of the same oc-vertex. These loops preserve the orders of the children of the oc-vertices.

Figure 3.23(a) shows a subgraph of the constraint tree and graph from Fig. 3.22 and Fig. 3.23(b) shows the corresponding labeled vertex-exchange graph (assuming all edges are '+'-labeled). The fat edges are added due to the two mc-vertices, m_1 and m_2, and the dotted loops are added due to the oc-vertex, o. The dummy vertex x is added to make the constraint tree proper.

The level planarity testing and embedding algorithms remain unchanged. If the level planarity test succeeds and we produce a level planar embedding, then we can contract G' and ϕ' back to G and ϕ respectively to remove the constraint trees from the level planar embedding.

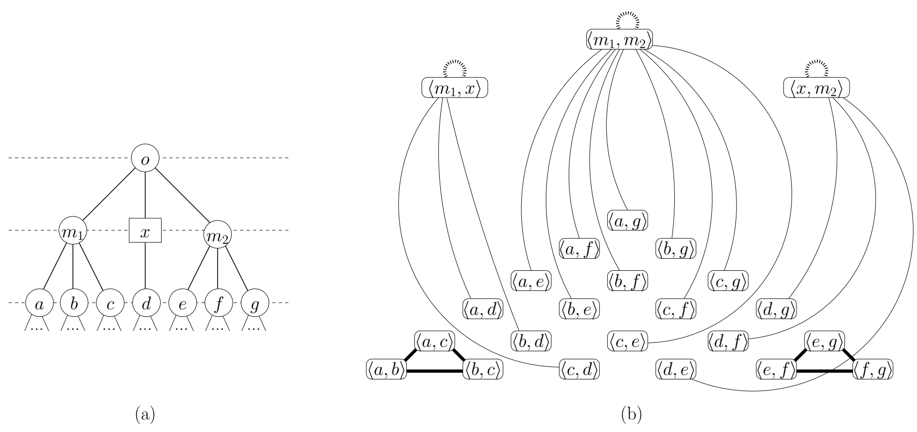


Figure 3.23: (a) A subgraph of the constraint tree and graph from Fig. 3.22. (b) The corresponding (labeled-) vertex-exchange graph (assuming all edges are '+'-labeled). The fat edges are added due to the two mc-vertices and the dotted loops are added due to the oc-vertex.

3.3.3 Multi-Level Crossing Minimization

We state an open problem, taken from a survey that appeared in the proceedings of a recent graph drawing conference (Brandenburg et al., 2004): *(Multi-level (or global) crossing minimization for the Sugiyama framework.)* Design an effective heuristic that can perform crossing minimization for a leveled graph over more than two levels at a time.

More formally, the *k-level crossing minimization problem* can be stated as:

Problem 3.3.1 (k-LEVEL-CROSSING-MIN). *Given a graph G with leveling ϕ, find a k-leveled drawing of G in which the number of edge crossings (the number of pairs of independent edges that violate Condition 1 of Definition 2.3.3) is a minimum.*

There exist many heuristics for k-LEVEL-CROSSING-MIN. These will be presented in the next chapter in §4.1. However, for the moment we will use the signed nature of the labeled vertex-exchange graph to apply another family of algorithms to k-LEVEL-CROSSING-MIN, namely, those algorithms that can solve the *maximum balanced subgraph problem*, MAX-BALANCED-SUBGRAPH.

Problem 3.3.2 (MAX-BALANCED-SUBGRAPH). *Given a graph G*

with signature λ find a maximum subgraph of G that is balanced.

To apply these algorithms, we need the following theorem:

Theorem 3.3.3. *Given a graph G with leveling ϕ and an initial leveled embedding $\mathcal{L}$ of G, let M be the maximum balanced subgraph of $\mathcal{LVE}(G, \phi, \mathcal{L})$. Now, by considering only those edges in M, and performing the level planarity testing and embedding algorithms (Algorithms 1 and 2), we can solve k-LEVEL-CROSSING-MIN.*

Proof: Maximizing the size of the balanced subgraph means maximizing the number of '+'-labeled edges. But maximizing this number requires minimizing the number of '–'-labeled edges. Each such edge represents a pair of edges in the leveled graph G. By 'ignoring' a minimum number of these pairs and removing all edge crossings from the remainder of the graph, we are ignoring a minimum number of edge crossings.

□

Fortunately, there already exists a selection of algorithms, both exact and approximate, for tackling MAX-BALANCED-SUBGRAPH, each of which can be applied to k-LEVEL-CROSSING-MIN.

Exact Algorithms

Dujmović and Whitesides (2004) provide an $\mathcal{O}(\phi^c n^2)$ FPT algorithm for the 2-level crossing minimization problem with the order of the vertices on one level fixed where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ is the golden ratio and the parameter c is the number of allowed edge crossings. Their algorithm is based on a *bounded search tree*. This involves exploring some search tree whose size depends only on the parameter c. The time spent at each vertex of the search tree must be polynomial in the size of the input. Then, for each fixed c, the size of the search space is constant and the running-time of the algorithm is polynomial.

Their algorithm hinges on the following observation. They partition all possible pairs of vertices on the free level into three subsets:

1. *trivially suited pairs* whose vertices have degree one;
2. *non-trivially suited pairs* whose vertices can be ordered so that their incident edges do not cross each other and at least one of the vertices has degree greater than one;
3. *unsuited pairs* (all others).

Trivially and non-trivially suited pairs have a *natural order*, *i.e.* an order in which their incident edges do not cross each other. In fact, for trivially suited pairs, either order is natural. Their observation can then be stated:

Theorem 3.3.4. *(Dujmović and Whitesides, 2004)*

Any 2-leveled drawing of a leveled graph G with the order of the vertices on one level fixed that has the minimum number of edge crossings, has all suited pairs on the free level in their natural order.

The order of the non-trivially suited pairs (and their transitive closure) is then known. A search tree of depth at most c is traversed to try both orders of each remaining unsuited pair. If c is set to some upper bound on the number of edge crossings, for example the upper bound of Nagamochi (2004, 2005a,b), then an optimal solution can be found. The running-time has subsequently been improved to $\mathcal{O}(1.4656^c + n^2)$ (Dujmović et al., to appear). However, these methods work only for two levels when one of the levels is fixed, and are intended for use as part of a level-by-level sweep heuristic (see §4.1).

On the other hand, there exists an FPT algorithm that can solve MAX-BALANCED-SUBGRAPH (Hüffner et al., 2007), thereby solving k-LEVEL-CROSSING-MIN over all levels simultaneously. Figure 3.24 depicts an 8-leveled drawing of a leveled graph G with 96 vertices and 110 edges altogether. This graph was part of a randomly generated dataset (Kuusik, 2000) used to experiment with Integer Linear Programming (ILP) approaches to k-LEVEL-CROSSING-MIN. Our drawing was produced by solving MAX-BALANCED-SUBGRAPH on $\mathcal{LVE}(G, \phi, \mathcal{L})$ using an FPT algorithm by

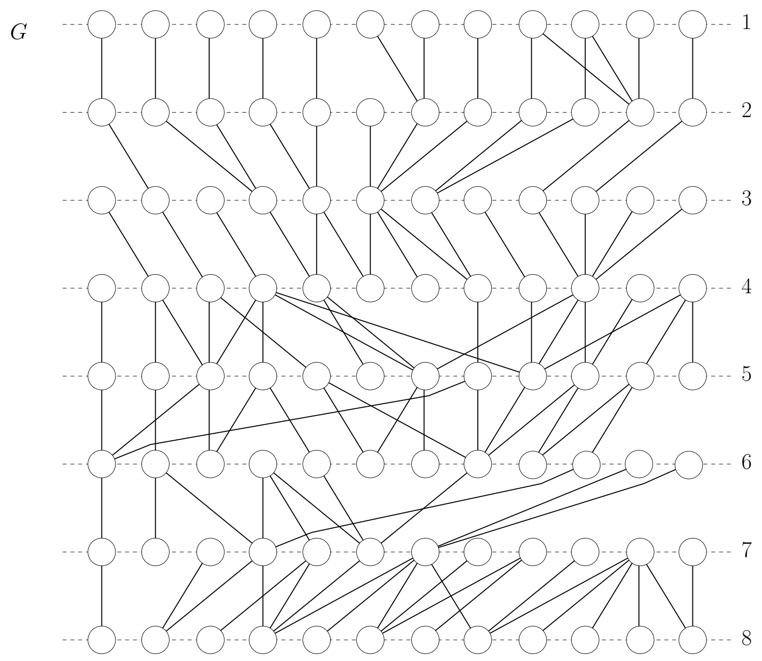


Figure 3.24: An 8-leveled drawing of a randomly generated leveled graph with the minimum number of edge crossings computed by solving MAX-BALANCED-SUBGRAPH.

Hüffner et al. (2007) and then applying Algorithms 1 and 2. Our drawing has the minimum number of edge crossings (31 in total), as can be verified from the ILP results (Kuusik, 2000) on the same graph. We also note that Barahona and Mahjoub (1988) have studied an ILP formulation of MAX-BALANCED-SUBGRAPH.

Approximation Algorithms

For MAX-BALANCED-SUBGRAPH, the best known approximation algorithm was motivated by a problem in biological networks (DasGupta et al., 2007). A useful approach to the analysis of such networks is to decompose a given system into an interconnection of simpler 'monotone' systems. This involves the removal of a minimum number of 'inconsistent' connections from otherwise 'consistent' components' and can be formulated as MAX-BALANCED-SUBGRAPH. DasGupta et al. (2007) provide a constant ratio inapproximability result: for some constant $\epsilon > 0$, it is not possible to approximate MAX-BALANCED-SUBGRAPH in polynomial time to within an approximation ratio of $1 - \epsilon$ and $1 + \epsilon$, unless $\mathsf{P} = \mathsf{NP}$. However, they also provide a polynomial time α-approximation algorithm where $\alpha = 0.87856$ is the approximation factor for MAX-CUT (Goemans and Williamson, 1995). By Theorem 3.3.3, this approximation algorithm can also tackle k-LEVEL-CROSSING-MIN.

Chapter 4

Crossing Minimization and Maximum Planarization

The difficulty in finding a k-leveled drawing of a graph that minimizes the number of edge crossings (Eades and Wormald, 1994; Muñoz et al., 2003) or maximizes a level planar subgraph (Eades and Whitesides, 1994) has long been known. In this chapter, we examine the simpler case where the graph is a tree. Conventional drawing algorithms for trees (Reingold and Tilford, 1981; Walker, 1990; Buchheim et al., 2006) assume the presence of a single root and always result in drawings without any edge crossings. However, if the tree has a leveling, there may be many vertices that warrant root status and some edge crossings become unavoidable. As we will see, a level-by-level approach to crossing minimization and maximum planarization becomes much easier for trees. However, the crossing minimization problem again becomes difficult when we consider multiple levels together.

The k-level crossing minimization problem for graphs, for both $k = 2$ and $k = 2$ with the order of the vertices on one level fixed is NP-Hard (Eades and Wormald, 1994; Muñoz et al., 2003). If $k = 2, 3$ then the problem for a large class of graphs (namely, those graphs in which $|E(G)| > (2+\epsilon)|V(G)|$ for some $\epsilon > 0$, every vertex in the second level must be adjacent to at least one vertex on each of the extreme levels and the maximum degree of any vertex in the middle level is bounded by a constant) can be approximated to within a factor of $\mathcal{O}(\log n)$ from the optimal in polynomial time (Shahrokhi and Vrťo, 2000; Shahrokhi et al., 2001). If $k = 2$ and the graph is a tree then it can be solved in $|V(G)|^{\lambda}$ time where $\lambda \approx 1.6$ (Shiloach, 1979; Chung, 1984; Shahrokhi et al., 2001). If $k = 2$ and the graph is a complete binary tree then it can be solved in linear time (Albacea, 2006). We consider the cases where the graph is a tree, $k = 2$ and the order of the vertices on one level may or may not be fixed and the case where the graph is a tree and $k > 2$ (see Table 4.1).

The k-level maximum planar subgraph problem for graphs, for $k = 2$ and all the vertices in each subset having degree two and three respectively, is NP-Hard (Eades and Whitesides, 1994). We consider the cases where the graph is a tree, $k = 2$ and the order of the vertices on one level may or may not be fixed (see Table 4.2). We also show that the cases where the graph is unicyclic or *series-parallel* with degree-one vertices attached and $k = 2$ can also be solved in linear time by solving the associated Hamiltonian cycle completion problem. A series-parallel graph is a graph (possible with multiple edges) with two distinguished vertices s and t that can be turned into K_2 (the graph with two vertices a single edge between them) by a sequence of the following operations: (i) the replacement of a pair of multiple edges with a single edge that connects their common endpoints and (ii) the replacement of a pair of edges incident to a vertex of degree two other than s or t with a single edge.

This chapter is organized as follows. In §4.1 we present some previous work with regard to heuristics and exact methods for solving the k-level crossing minimization and maximum planarization problems. In §4.2 we examine the crossing minimization problem for trees when $k = 2$ and for arbitrary k. In §4.3 we present some analogous results for the maximum level planar subgraph problem. The cases where the graph is unicyclic or series-parallel with degree-one vertices attached and $k = 2$ is treated in §4.3.1. Our contributions in this chapter include the 'boxed' complexity results in Tables 4.1 and 4.2, Corollary 4.2.3, the connection between the *Hamiltonian cycle completion problem* and the *maximum level planar subgraph* problem and the *planarizing*

	$k = 2$	$k = 2$, one level fixed	Arbitrary k
Graphs	NP-Hard[a]	NP-Hard[b]	NP-Hard
Trees	P[c]	$\boxed{\text{P}}$	$\boxed{\text{NP-Hard}}$

[a] Garey and Johnson (1983)
[b] Eades and Wormald (1994); Muñoz et al. (2003)
[c] Shahrokhi et al. (2001); Chung (1984); Shiloach (1979)

Table 4.1: The k-level crossing minimization problem. The results in boxes are provided by this chapter.

	$k = 2$	$k = 2$, one level fixed	Arbitrary k
Graphs	NP-Hard[d]	NP-Hard[e]	NP-Hard
Trees	P[f]	$\boxed{\text{P}}$	?

[d] Eades and Whitesides (1994); Tomii et al. (1977)
[e] Eades and Whitesides (1994)
[f] Shahrokhi et al. (2001)

Table 4.2: The k-level maximum level planar subgraph problem. The result in the box is provided by this chapter.

penalty digraph.

4.1 Previous Work

The problem of minimizing the number of edge crossings in a 2-leveled drawing of a graph was first introduced by Harary and Schwenk (1971, 1972). Since then a multitude of heuristics and exact methods have appeared in the literature. Some of the heuristics include the iterated barycenter heuristic (Jünger and Mutzel, 1997), a heuristic based on a Fiedler vector (Newton et al., 2003), simulated annealing (Newton et al., 2003) and metaheuristics (Martì and Laguna, 2003). Exact methods include the branch-and-bound algorithm of Valls et al. (1996) and the branch-and-cut algorithms of Jünger and Mutzel (1997) and Zheng and Buchheim (2007).

If we are required to keep the order of the vertices on one of the levels fixed then the following heuristics apply: the greedy insertion, greedy switching and split heuristics (Eades and Kelly, 1986), the median heuristic (Eades and Wormald, 1994), the assignment heuristic (Catarci, 1995), the stochastic heuristic (Dresbach, 1994), metaheuristics (Mäkinen and Sieranta, 1994) and, if the maximum degree of the any vertex on the free level is at most four, the recent heuristic of Yamaguchi and Sugimoto (1999). One of the most important heuristics is the barycenter heuristic (Sugiyama et al., 1981) which finds an $\mathcal{O}(\sqrt{n})$-approximation solution or a $(d-1)$-approximation solution, where d is the maximum degree of any vertex on the free level (see Y-Li and Sallmann (2002) for the analysis). There has been extensive computational experimentation involving several of these heuristics (Mäkinen, 1990; Jünger and Mutzel, 1997). Jünger and Mutzel (1997) reported that most of the heuristics performed well, providing crossing numbers close to a simple lower bound. They also compared the heuristics to an exact branch-and-cut algorithm.

There are several approaches to tackling the k-level crossing minimization problem. The *level-by-level sweep* (Sugiyama et al., 1981; Gansner et al., 1993; Sander, 1995) is the most widely known. For a level j, the vertices on level $j-1$ and $j+1$ (if they exist) are held fixed while the vertices on level j are permuted. This can be successively applied for increasing and decreasing values of j. Specific implementations differ in three ways: the heuristic, the sweeping direction, and the stopping criterion. With *sifting* (Matuszewski et al., 2000; Günther et al., 2001), the vertices on each level are ordered according to their degree and then each vertex is individually moved to its best possible position with all others remaining fixed. A crossing number matrix counts the crossings for each position. With *windows*, subsets of the k-leveled graph that match certain criteria are solved one at a time (Eschbach et al., 2003). There is also a choice of metaheuristics (Laguna et al., 1997; Utech et al., 1998; Kuntz et al., 2004, 2006). Exact methods are explicated by Jünger et al. (1998) and Kuusik (2000). Of course, to this list we can now add our own approach of §3.3.3.

The maximum level planar subgraph problem has received comparatively less study. For the 2-level maximum level planar subgraph problem we can use the exact branch-and-cut algorithm of Mutzel (1997, 2001a) or the FPT algorithms of Dujmović et al. (2006) and Fernau (2005). Suderman and Whitesides (2005) have experimented with the branch-and-cut algorithm (Mutzel, 1997, 2001a) and the FPT algorithm (Dujmović et al., 2006) reporting comparable results from both. If we are required to keep the order of the vertices on one of the levels fixed then branch-and-cut (Mutzel and Weiskircher, 1998) and FPT algorithms (Dujmović et al., 2006; Fernau, 2005) are also available to us. Finally, Kuusik (2000) has provided a branch-and-cut algorithm for the k-level maximum level planar subgraph problem.

4.2 Crossing Minimization for Leveled Trees

In this section we examine the problem of finding a k-leveled drawing of a tree with the minimum number of edge crossings. In §4.2.1 we review a result of Shahrokhi et al. (2001) that uses a *minimum linear arrangement* of a tree to solve the problem when $k = 2$. In §4.2.2 we show that the problem can also be solved in polynomial running-time when the order of the vertices on one level is fixed but in §4.2.3 we prove the NP-Hardness of the problem for arbitrary k.

4.2.1 2-Leveled Trees

A bijective mapping $\pi : V(T) \to \{1, 2, \dots, n\}$ of the vertices of a tree T is called a *linear arrangement* π of T. The *cost* of a linear arrangement π, $\texttt{cost}_\pi(T)$, is $\sum_{\{u,v\}\in E} |\pi(u) - \pi(v)|$. The *cost* of T, $\texttt{cost}(T)$, is the minimum value of $\texttt{cost}_\pi(T)$ as π ranges over all possible linear arrangements of T. A linear arrangement π is *minimum* if $\texttt{cost}_\pi(T) = \texttt{cost}(T)$.

Theorem 4.2.1. *(Shahrokhi et al., 2001)*

The 2-leveled drawing of T with leveling $\phi : V(T) \to \{1, 2\}$ obtained from a minimum linear arrangement π by setting $u <_j v$ if and only if $\pi(u) < \pi(v)$, $u, v \in \phi^{-1}(j), 1 \le j \le 2$ has the minimum number of edge crossings. In fact, the number of edge crossings is equal to $\texttt{cost}(T) - n + 1 - \sum_{v\in V(T)} \lfloor \frac{deg(v)}{2} \rfloor \lceil \frac{deg(v)-2}{2} \rceil$.

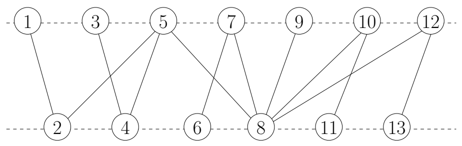


Figure 4.1: A 2-leveled drawing of a tree with the minimum number of edge crossings obtained from a minimum linear arrangement. The vertices are labeled by their positions in the minimum linear arrangement.

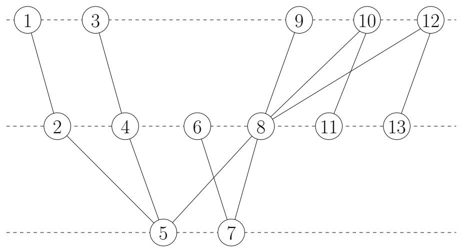


Figure 4.2: A 3-leveled drawing of a tree obtained from a minimum linear arrangement. Note that this drawing does not have the minimum number of edge crossings. The vertices are labeled by their positions in the minimum linear arrangement.

Therefore any algorithm that can find a minimum linear arrangement of a tree can also find a 2-leveled drawing of a tree with the minimum number of edge crossings (Shiloach, 1979; Chung, 1984). It is easy to see that Theorem 4.2.1 does not hold for $k > 2$ levels (see Figs. 4.1 and 4.2).

4.2.2 2-Leveled Trees with One Level Fixed

Suppose we are given a graph G, a leveling $\phi : V(G) \to \{1, 2\}$, a total order $<_2$ of the vertices in $\phi^{-1}(2)$ and some initial 2-leveled drawing of G that respects $<_2$. We define $\mathtt{cr}(u, v), \forall u, v \in \phi^{-1}(1)$ to be the number of edge crossings produced by edges incident with u and v when u is placed to the right of v. We define the *crossing penalty digraph* of G to be the directed graph (possibly with multiple edges) CP_G with vertex set $\phi^{-1}(1)$ and $\mathtt{cr}(u, v) - \mathtt{cr}(v, u)$ directed edges from u to v if and only if $\mathtt{cr}(u, v) > \mathtt{cr}(v, u)$. A *feedback arc set (FAS)* of a directed graph is a set of edges whose reversal makes the directed graph acyclic. A *minimum FAS* is a minimal such set. For our purposes, a minimum FAS of CP_G is a minimum number of directed edges in $E(CP_G)$ whose reversal makes CP_G acyclic. The utility of the crossing penalty digraph has been

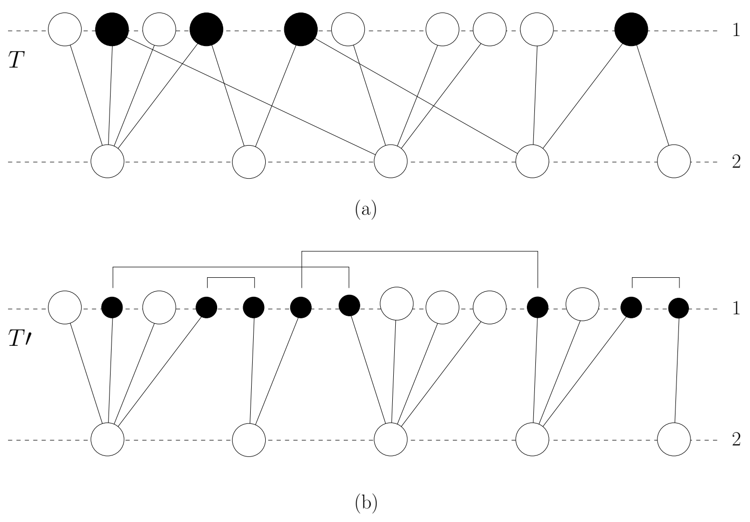


Figure 4.3: (a) The bold vertices in T must be split. (b) The stars in T' are placed 'side-by-side'. The newly split vertices are indicated by the joining lines above level 1.

established by the following theorem:

Theorem 4.2.2. *(Sugiyama et al., 1981; Demestrescu and Finocchi, 2001)*

Finding a 2-leveled drawing of G that respects $<_2$ with the minimum number of edge crossings is equivalent to finding a minimum feedback arc set in CP_G. A topological order of $V(CP_G)$ with a minimum feedback arc set reversed gives the required order of the vertices on the free level of G.

For the special case of trees, we prove the corollary:

Corollary 4.2.3. *Finding a 2-leveled drawing of a tree T that respects $<_2$ with the minimum number of edge crossings is equivalent to finding a topological order of $V(CP_T)$ since CP_T is always a DAG.*

Proof: For a topological order of $V(CP_T)$ to exist, CP_T must be acyclic. Split every vertex $u \in \phi^{-1}(1)$ into $\mathtt{deg}(u)$ new vertices, each joined to exactly one of u's former neighbors. Call this new forest of stars T'. By placing the stars 'side-by-side' we can derive a new leveling ϕ' and 2-leveled drawing of T' that respects $<_2$ with no edge crossings (see Fig. 4.3(a)–(b)). At this stage $CP_{T'}$ is obviously acyclic.

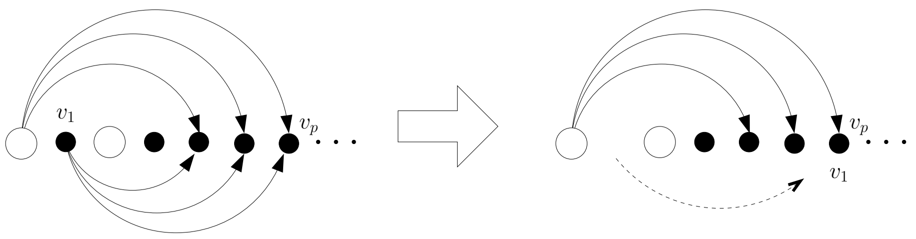


Figure 4.4: Contracting v_1 and v_p will not result in a 2-cycle in $CP_{T'}$ since the two directed edges will cancel each other out.

Choose a maximum subset of vertices $v_1, \ldots, v_p \in V(CP_{T'})$ (ordered left-to-right in the new 2-leveled drawing) that were split from the same vertex in T and hence belong to p different components of T'. Contract the vertices and place the newly contracted vertex in the position of v_p. Update $CP_{T'}$. If the p components occur next to each other in T' then the directed edge(s) between $v_1, \ldots, v_p$ in $CP_{T'}$ disappear and the directed edges that started (resp. ended) at $v_1, \ldots, v_p$ now start (resp. end) at the newly contracted vertex, thus preserving the acyclicity of $CP_{T'}$. On the other hand, there may be other components in between the p components in T'. For example, suppose $(v_1, u), (u, v_2) \in E(CP_{T'})$ where u does not belong to any of the p components and we need to contract v_1 and v_2 to the same vertex in T'. Observe that this will not result in a 2-cycle (a directed cycle with two edges) in $CP_{T'}$ since the two directed edges will cancel each other out, *i.e.* they will contribute at least one crossing no matter how we order them (see Fig. 4.4). By repeated contractions we can merge all the split vertices back to form T and observe that CP_T must be acyclic.

□

4.2.3 k-Leveled Trees

We state the decision problem k-LEVEL-CROSSING-MIN for trees as follows. Given a tree T, a leveling $\phi : V(T) \rightarrow \{1, 2, \ldots, k\}$ and a positive integer K, is $\texttt{kcr}(T, \phi) \leq K$?

Theorem 4.2.4. k-LEVEL-CROSSING-MIN *for trees is* NP*-Hard.*

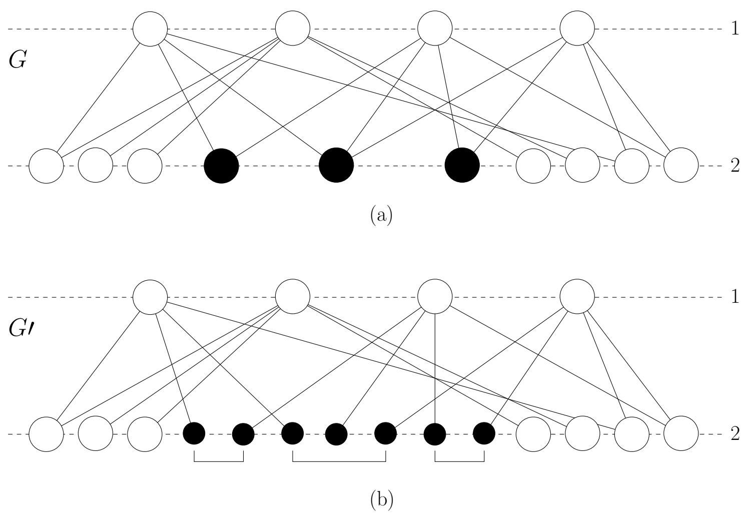


Figure 4.5: (a) A graph G with leveling $\phi : V(G) \to \{1, 2\}$. (b) Splitting non-cut vertices with degree greater than one (the filled vertices) along level 2 results in G'.

Proof: Clearly, k-LEVEL-CROSSING-MIN is in NP. We reduce the decision version of the 2-level crossing minimization problem for general graphs when the order of the vertices on one level is fixed to k-LEVEL-CROSSING-MIN for trees.

Let G be a graph with leveling $\phi : V(G) \to \{1, 2\}$ and let $<_2$ be a total order of $\phi^{-1}(2)$. Choose a non-cut vertex $u \in \phi^{-1}(2)$ with $\mathtt{deg}(u) > 1$ and split u into $\mathtt{deg}(u)$ new vertices, each joined to exactly one of u's former neighbors. Repeat this step until G is a tree. We call this new graph G'. Let ϕ' be a leveling of G' that maps the vertices in $V(G) \cap V(G')$ in the same way as ϕ and the vertices in $V(G') \setminus V(G)$ to level 2. Figure 4.5(a)–(b) exemplifies this process.

Let $<'_2$ be a partial order of $\phi'^{-1}(2)$ as follows:

1. If $u, v \in V(G) \cap V(G')$ then the relative order of u and v is the same as in $<_2$.

2. If $u \in V(G) \cap V(G')$ and $v \in V(G') \setminus V(G)$ then the relative order of u and v is the same as that of u and the vertex from which v was split in $<_2$.

3. If $u, v \in V(G') \setminus V(G)$ and u and v were split from different vertices then

the relative order of u and v is the same as that of the vertices from which u and v were split in $<_2$ respectively.

4. If $u, v \in V(G') \setminus V(G)$ and u and v were split from the same vertex then the vertices are not compared.

It can be observed that a 2-leveled drawing of G' that respects the partial order $<_2'$ with the minimum number of edge crossings corresponds to a 2-leveled drawing of G that respects the total order $<_2$ with the minimum number of edge crossings by contracting the vertices in $V(G')$ that were split from the same vertex in $V(G)$.

We augment G' as follows. Find a linear extension L of the partial order $<_2'$. Note that vertices that were split from the same vertex in $V(G)$ appear consecutively. Let B be an upper bound on the number of edge crossings in any 2-leveled drawing of G, *e.g.* m^2. Let $s = 1$ and start with u, the left-most vertex in L. Join u to a new path $u = u_1, \ldots, u_s = v$ and set the level of each new vertex u_i to $i + 1$. Join v to a new path $v = v_1, \ldots, v_{s+1} = w$ and set the level of each new vertex v_i to $s - i + 2$. Join every vertex along $u = u_1, \ldots, u_s = v = v_1, \ldots, v_{s+1} = w$ to $2B$ new leaves. Place B of these leaves to the level one less than that of the vertex to which they are joined and set the remaining B leaves to the level one greater than that of the vertex to which they are joined. Join v and w to a further ΔB^2 new leaves where Δ is the maximum degree of the vertices in $\phi^{-1}(2)$ and set their respective levels to one greater than the level of their root.

Repeat these steps for any successive vertices in L that were split from the same vertex in $V(G)$ as u. When no more successive vertices exist, increment s and move to u', the right-most vertex in L. Repeat the steps for u' and any predecessive vertices that were split from the same vertex in $V(G)$ as u'. Increment s and move to the left-most vertex in L that we have yet to process, and so on. Continue until all vertices in L have been processed. We call this new tree T with leveling ϕ_T. Figure 4.6 shows the augmentation applied to the tree in Fig. 4.5(b).

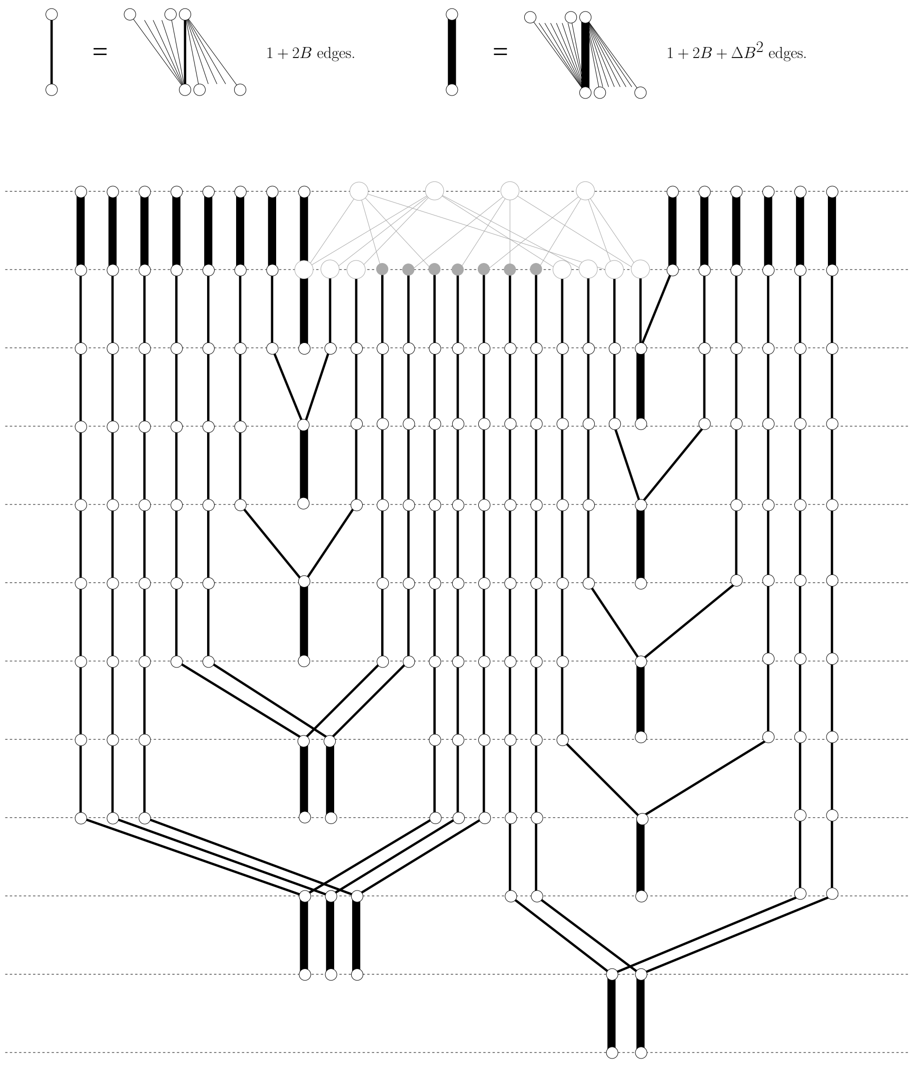


Figure 4.6: The augmentation applied to the tree in Fig. 4.5(b).

We have essentially augmented every vertex in $\phi'^{-1}(2)$ with a 'strut'. In any k-leveled drawing of T with the minimum number of edge crossings, struts whose corresponding vertices in $\phi'^{-1}(2)$ were not split from the same vertex in G do not cross, whereas struts whose corresponding vertices in $\phi'^{-1}(2)$ were split from the same vertex in G cross a minimum number of times as shown in Fig. 4.6. The struts enforce the partial order $<'_2$ (or its reverse) on $\phi'^{-1}(2)$.

Let $p = |\phi'^{-1}(2)|$. There are exactly p struts and each spans at most p levels. The number of leaves attached to each vertex on the strut is bounded by B and so the total size of the augmentation is polynomial in the size of G. It can also be observed that a k-leveled drawing of T with the minimum number of edge crossings corresponds to a 2-leveled drawing G' with the minimum number of edge crossings that respects the partial order $<'_2$ by simply removing the struts.

□

It is interesting to note that the proof of Theorem 4.2.4 also shows that the crossing minimization problem for 2-leveled graphs when the order of the vertices on one level is subject to some partial order is NP-Hard. This is in contrast to the results of the previous two subsections.

4.3 Maximum Level Planar Subgraph for Leveled Trees

In this section we examine the problem of finding a k-leveled drawing of a tree with a maximum level planar subgraph, *i.e.* a maximum subgraph with no edge crossings. In §4.3.1 we review some methods of solving the problem when $k = 2$. One of these, which shows an equivalence to the Hamiltonian cycle completion problem, appears to be new. In fact, it also solves the problem for unicyclic and series-parallel graphs with degree-one vertices attached. In §4.3.2 we show that the problem can also be solved when the order of the vertices on one level is fixed through the use of a *planarizing penalty digraph*.

4.3.1 2-Leveled Trees

We recall a well-known characterization of level planarity for 2-leveled graphs:

Theorem 4.3.1. *(Tomii et al., 1977)*

A graph G with leveling $\phi : V(G) \to \{1, 2\}$ is level planar if and only if G is a forest of caterpillars.

Tomii et al. (1977) have proposed an algorithm based on this characterization with cubic running-time to find a maximum level planar subgraph of a tree T with leveling $\phi : V(T) \to \{1, 2\}$. However, it has since been shown to be incorrect (Mutzel, 2001a). An outline for a correct but naïve dynamic programming algorithm based on this characterization is as follows. Choose some $r \in T' = T \setminus \{v \in V(T) | \mathtt{deg}(v) = 1\}$. In any maximum level planar subgraph of T, r must belong to the backbone of one its constituent caterpillars. Suppose the backbone is $(u, \ldots, r, \ldots, v)$. Note that both u and v may be r itself. There are $\mathcal{O}(n^2)$ choices for the $\{u, v\}$ pair. At least one of these must correspond to the backbone of a caterpillar in a maximum level planar subgraph of T. Remove each backbone in turn along with its incident edges from T'. We are left with a forest of (rooted) subtrees of T'. The maximum level planar subgraph is also maximal and level planar when restricted to these subtrees. In summary, the algorithm considers all possible backbones that could include r, solves the problem on the resultant subtrees and then puts together a maximum level planar subgraph to form a feasible solution. The best feasible solution (the one with the most edges) is then a maximum level planar subgraph of T.

Alternatively, and much more efficiently, we can find a minimal *path cover* of T', *i.e.* a minimal set of disjoint paths that cover all the vertices of T'. Each path corresponds to a backbone of a caterpillar in T and, since the number of paths is minimal, the caterpillars constitute a maximum level planar subgraph. Finding a minimal path cover of a graph (not just a tree) is equivalent to the *Hamiltonian cycle completion problem* (Goodman et al., 1975). The Hamiltonian cycle completion problem for a graph consists of finding a minimum number of edges which can be added to the graph to produce a Hamiltonian cycle. It is obviously NP-Hard since it contains the classical Hamiltonian cycle problem as a special case. However, the problem can be solved in linear time when restricted to several classes of graphs which include, amongst others, trees (Goodman et al., 1975), unicyclic graphs (Goodman et al., 1975) and series-parallel graphs (Takamizawa et al., 1982). Therefore, the maximum level planar subgraph problem when $k = 2$ can be solved in linear time for all graphs that fall into these classes once their degree-one vertices have been removed. Shahrokhi et al. (2001) have also provided an algorithm with linear running-time for the weighted version of the maximum level planar subgraph problem when $k = 2$ and the graph is a tree.

4.3.2 2-Leveled Trees with One Level Fixed

Suppose we have a tree T, a leveling $\phi : V(T) \to \{1, 2\}$ and a total order $<_2$ of the vertices in $\phi^{-1}(2)$. We define the *planarizing penalty digraph* of T as the directed graph PP_T with vertex set $E(T)$ and directed edges $(e, f) \in E(PP_T)$ if and only if:

1. e and f share a vertex on level 2; or
2. e and f share a vertex on level 1 and e's vertex on level 2 precedes f's vertex on level 2 in $<_2$; or
3. e and f are independent and e's vertex on level 2 precedes f's vertex on level 2 in $<_2$.

Observe that the edges added by (1) and (2) induce cliques and acyclic tournaments in PP_T respectively. This brings us to our theorem:

Theorem 4.3.2. *Finding a 2-leveled drawing of T that respects $<_2$ with a maximum level planar subgraph is equivalent to finding a longest path in PP_T that enters and leaves each induced acyclic tournament at most once.*

Proof: The edges added to PP_T as a result of (1) induce cliques. The vertices along any path through one of these cliques represent a subset of edges in T ordered left-to-right and incident with their shared vertex on level 2 that, on their own, are level planar. The edges added to PP_T as a result of (2) join the cliques together through acyclic tournaments. The graph with each clique contracted to a single vertex does not contain any directed cycles. The vertices along any path through this contracted graph represent a subset of edges in T ordered left-to-right and incident with their shared vertex on level 1 that, on their own, are also level planar. The edges added to PP_T as a result of (3) allow 'shortcuts' through PP_T. They do not add any directed cycles to PP_T (this can be seen from a topological order of the contracted cliques and noting that every edge added by (2) and (3) is directed from left to right). We restrict the paths through the entire graph to enter and leave each induced acyclic tournament at most once. Therefore, a path through the entire graph represents a subset of edges in T ordered left-to-right and incident with their corresponding vertices on either levels that is level planar. The converse is also true; any level planar subgraph in T has a corresponding path through PP_T that enters and leaves each induced acyclic tournament at most once. Each vertex in the path corresponds to one edge in the level planar subgraph, so a maximum level planar subgraph in T corresponds to a longest path in PP_T that enters and leaves each induced acyclic tournament at most once.

□

We can combine (2) and (3) to get 'add a directed edge from e to f if and only if e's vertex on level 2 precedes f's vertex on level 2 in $<_2$'. However, separating them into two cases makes the proof and their 'reason for being' somewhat clearer. It also makes it easier to identity the induced acyclic tournaments.

A longest path in a DAG can be found in linear time using dynamic programming. However, PP_T may contain directed cycles and we require a longest path that enters and leaves each induced acyclic tournament at most once. To this end, we note that the directed cycles occur only within the induced cliques and the requirement with regard to the tournaments can also be handled by dynamic programming.

Consider the 2-leveled drawing of a tree T in Fig. 4.7 for example. The vertices on level 2 are subject to a total order $<_2$. An abridged planarizing penalty digraph PP_T is shown in Fig. 4.8. Edges added by (1) are solid, by (2) are dashed and by (3) are omitted for clarity. The edges added by (3) join every vertex to every vertex below it, if they were not already joined by edges from (2). It is also convenient to assume the presence of a dummy source vertex s at the top with directed edges joining it to every other vertex. This provides a convenient place to start the traversal of the graph and ensures that the length of the longest path will equal the number of edges in the maximum level planar subgraph.

At each vertex v we keep track of several counters: the length of the longest path to the vertex found so far, $p_0[v]$, and the lengths of the longest paths to the vertex that do not use each of the induced acyclic tournaments found so far, $p_1[v], p_2[v], \ldots, p_t[v]$. For our example $t = 4$. We arbitrarily label the induced acyclic tournaments to match these indices. In our example, 1 is the acyclic tournament induced by b, d and e, 2 by f and g, 3 by i and k and 4 by j and l.

All counters are initialized to zero. We find a topological order of $V(PP_T) \cup \{s\}$ treating each induced clique as a single group of vertices that appear together. A topological order for our example is $s, a, b, c, d, f, e, g, h, i, j, k, l$. Note that f appears before e as a result of the order of their end points in $<_2$. We process the vertices in the topological order from left-to-right. For a vertex v, we examine each outgoing neighbor $w \in \mathcal{N}^+(v)$. If v is the first vertex to be visited in an induced clique then we save a copy of the counters of each vertex in the clique. These copies are denoted by $p_0^*[u], p_1^*[u], \ldots, p_t^*[u]$ for each vertex u in the clique. There are now two cases: (1) w is also in the clique: We set $p_0[w] = \max\{p_0^*[w], p_0[v]^* + 1\}$. For each $i > 0$ we set $p_i[w] = \max\{p_i[w]^*, p_i[v]^* + 1\}$. The only exception to this occurs when v is

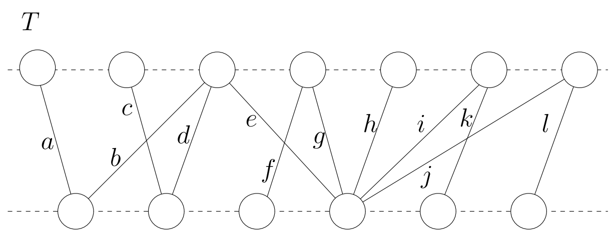


Figure 4.7: A 2-leveled drawing of a tree with labeled edges.

part of the induced acyclic tournament i and w is not. Here, we leave $p_i[w]$ as it is. (2) w is not in this clique: We set $p_0[w] = \max\{p_0[w], p_0[v] + 1\}$. For each $i > 0$ we set $p_i[w] = \max\{p_i[w], p_i[v] + 1\}$. The only exception to this occurs when v is part of the induced acyclic tournament i and w is not. Here, we leave $p_i[w]$ as it is.

Starting at s, we update all the counters to 1 since there is an edge from s to every other vertex. Next we visit a and update all the counters of all vertices after a in the topological order to 2. Then we visit b, which is part of the acyclic tournament 1. This updates a's counters as follows: p_1 remains the same while the others are incremented to 2. For all other vertices after b in the topological order, their p_1 counters remain the same while all others are incremented to 3. This continues to the end of the topological order.

When this process has concluded the counter with the greatest entry is the length of the longest path in PP_T that enters and leaves each induced acyclic tournament at most once. We can use a system of 'pointers' to trace this path back to s. In our example, a longest path $s, a, b, d, e, g, h, i, j, l$ constitutes a maximum level planar subgraph.

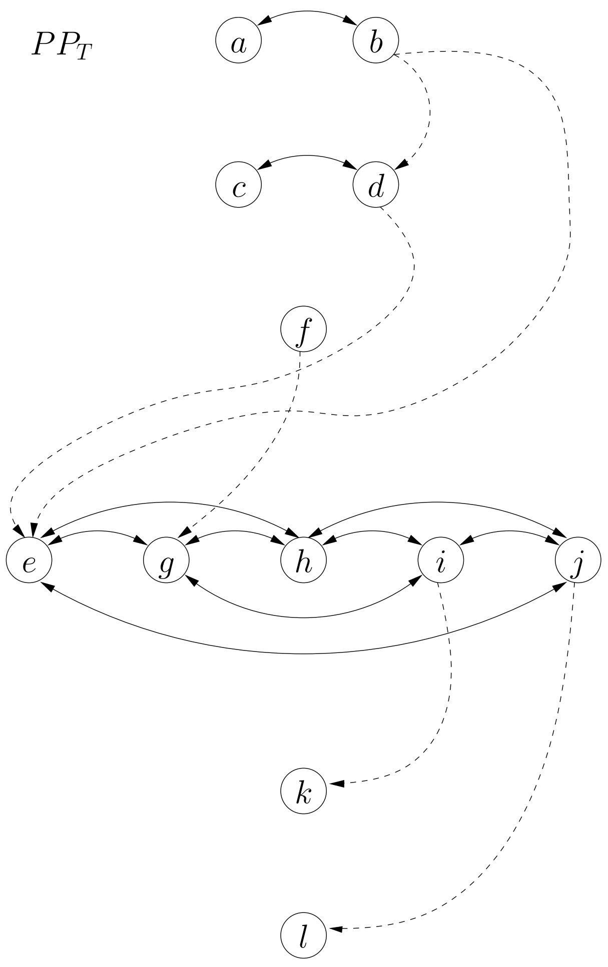


Figure 4.8: The planarizing penalty digraph PP_T for the tree in Fig. 4.7. Edges added by (1) are solid, by (2) are dashed and by (3) are omitted for clarity. The edges added by (3) join every vertex to every vertex below it, if they were not already joined by edges from (2).

Chapter 5

Ensuring Level Planarity

A popular strategy when drawing graphs is to extract a spanning tree, draw it using some simpler algorithm and then handle the remaining edges. When the graph is undirected this spanning tree is a free tree and we can nominate some vertex as the root (Botafogo et al., 1992), draw the spanning tree using the simpler tree drawing algorithms (Reingold and Tilford, 1981; Walker, 1990; Buchheim et al., 2006) and add in the remaining edges. However, in the directed case the spanning tree is not necessarily a directed tree with one single source; it is a *tree DAG* (a directed graph without any cycles, directed or undirected) with potentially multiple sources. It is inappropriate to choose one of these vertices arbitrarily as the root and so the simpler tree drawing algorithms are not suitable.

We propose finding a *significant spanning tree DAG T* using either domain-specific (*e.g.* edge weights associated with the graph) or graph-theoretic knowledge. We find a leveling of the graph that is level planar with respect to T by inserting a small number of dummy vertices and restrict the permutations of the vertices on each level to those that constitute a level planar embedding of T. In this way we ensure that any edge crossings in the final drawing do not involve two significant edges. We use a globally oriented Fiedler vector to choose permutations of the vertices on each level that reduce the number of edge crossings between the remaining edges. Our main contributions are: a proof that minimizing the number of dummy vertices, even in a simple 'base case', is NP-Hard, a heuristic method that runs in linear time and a demonstration of how this approach can be used to draw entire directed graphs.

Our approach can be considered a variation of the popular Sugiyama framework (Sugiyama et al., 1981). This framework temporarily removes any directed cycles by reversing a small number of edges, creates a proper leveling, permutes the vertices on each level to reduce the number of edge crossings and balances the layout. An alternative goal when permuting the vertices on each level is to maximize the size of a level planar subgraph. For two levels and two levels with the order of the vertices on one level fixed, both the crossing minimization and maximum level planar subgraph problems are NP-Hard (Eades and Wormald, 1994; Eades and Whitesides, 1994) and so heuristics are generally employed. Once we have computed a Fiedler vector, our method efficiently combines the leveling and crossing minimization steps in linear time. We particularly favor our method when we want a spanning tree DAG of the directed graph to be clear and apparent from the drawing.

This chapter is organized as follows. We begin by showing how to choose a significant spanning tree DAG T in §5.1. In §5.2 we present a recursive algorithm that ensures the level planarity of T by inserting a small number of dummy vertices. This also implies a simple embedding algorithm. In §5.3 we describe a crossing minimization heuristic based on a Fiedler vector and show how it can be combined with the previous step. §5.4 brings all the parts together to draw the dependency relationship between packages related to the Python programming language.

The results of this chapter have been reported previously (Harrigan and Healy, 2007, 2008b).

5.1 Choosing a Significant Spanning Tree DAG

Every tree DAG T (a directed graph with no cycles, directed or undirected) has a leveling ϕ such that all edges are directed uniformly. If T with leveling ϕ

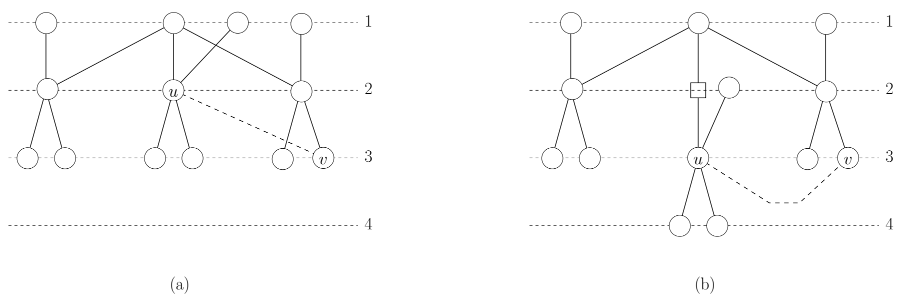


Figure 5.1: A non-level planar and level planar drawing of spanning tree DAGs (solid edges). All edges point downwards.

is not level planar then we can make it so by inserting dummy vertices along appropriate edges (see the spanning tree DAGs in Fig. 5.1). Accomplishing this with a small number of dummy vertices is the main concern of §5.2.

Let G be a directed graph. We assume the graph is connected; otherwise we handle each component separately. We seek a significant spanning tree DAG T of G, *i.e.* one in which the edges in $E(T)$ are deemed more significant than those in $E(G) \setminus E(T)$. This can be done using domain-specific knowledge or any of the numerous methods for measuring significance (*e.g.* spanning tree algorithms (Jungnickel, 1999, Chap. 2) and structural indices (Koschützki et al., 2005)). Recently, Lehmann and Kottler (2007) combined a number of measures to find a spanning tree or 'backbone' with a very low average spanner property for the non-tree edges and used it as input to an undirected tree drawing algorithm. The method we choose is to weight each edge $(u, v) \in E(G)$ by $(\vec{x}_u - \vec{x}_v)^2$ where $\vec{x}$ is a Fiedler vector of $\mathcal{L}(G)$ and $\vec{x}_u$ denotes the entry associated with vertex u. T is then a minimum weighted spanning tree DAG of G.

Our choice of T is primarily due to the fact that we will be re-using $\vec{x}$ later. However, we give two reasons to justify T as being a somewhat significant spanning tree DAG. Firstly, consider the problem of embedding G in the line so that all edge lengths are kept short. If the location of $v \in V(G)$ in the line is $\vec{x'}_v$ then we want to minimize $\sum_{(u,v)\in E(G)} (\vec{x'}_u - \vec{x'}_v)^2$ subject to $\vec{x'}\vec{x'}^T = \vec{1}$ (to avoid the trivial solution of setting $\vec{x'} = \vec{0}$) and $\vec{x'} \perp \vec{1}$. It turns out that a solution is precisely $\vec{x}$. Secondly, $\vec{x}_u$ can be interpreted as a measure of u's 'degreenormalized' *eigenvector centrality* (Bonacich, 1972) with $E(T)$ comprising the edges that propagate most of this centrality through G. Eigenvector centrality is a measure of the significance of a vertex in a graph. It assigns relative values to all vertices based on the principle that incident edges from high-value vertices contribute more to the value of the vertex in question than incident edges from low-value vertices (Koschützki et al., 2005).

However, since the choice of a meaningful spanning tree is crucial, our method is more suited to graphs whose edges are weighted *a priori* by domain-specific knowledge.

5.2 Ensuring Level Planarity of a Leveled Tree

Level planarity can be tested in $\mathcal{O}(n)$ time using the PQ-tree data structure (Heath and Pemmaraju, 1999; Jünger et al., 1999) or in $\mathcal{O}(n^2)$ time using the simpler vertex-exchange graph (Healy and Kuusik, 2004; Harrigan and Healy, 2008a). However, extending either method to ensure level planarity for the special case of tree DAGs by inserting a small number of dummy vertices is not obvious.

Our algorithm `makeTreeDAGLevelPlanar` is based on the recursive nature of a tree DAG. It processes a tree DAG T with leveling ϕ by recursively decomposing T into smaller tree DAGs with fewer vertices of in-degree greater than one. It computes a matrix $M(T)$ at each step where each column of $M(T)$ represents the restrictions imposed on the level planarity of T by each smaller tree DAG.

5.2.1 The Base Case

Let T be a tree DAG with leveling ϕ and $\mathcal{V} = \{v \in V(T) : \texttt{inDeg}(v) > 1\}$. We assume $|\mathcal{V}| > 0$ since otherwise T with ϕ is trivially level planar. In the base case $|\mathcal{V}| = 1$ and we let this vertex be r. The outgoing and incoming neighbors of r, $\mathcal{N}^+(r)$ and $\mathcal{N}^-(r)$, join r to the roots and vertices of distinct directed trees respectively (see Fig. 5.2). We use the notation T_v to refer to

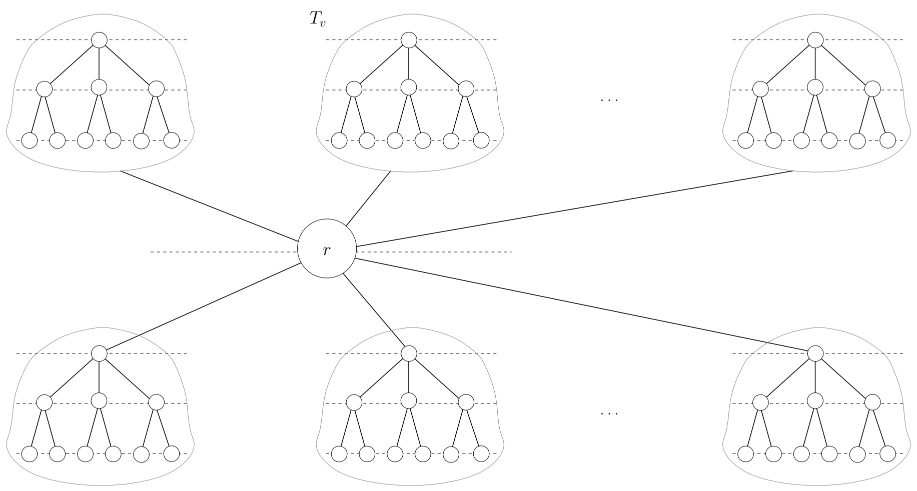


Figure 5.2: The base case has exactly one vertex with in-degree greater than one.

the maximal subtree that is joined to r through some $v \in \mathcal{N}^+(r) \cup \mathcal{N}^-(r)$. To ensure level planarity, we need only insert dummy vertices along edges joining $\mathcal{N}^-(r)$ to r.

We proceed by populating a matrix $M(T)$. For each $v \in \mathcal{N}^-(r)$ we add a column to $M(T)$ as follows. We visit each vertex v' along the (undirected) path from v to the root of its respective tree T_v. If there exists some vertex w that is a descendant (reachable by a ϕ-monotonic path) of v' in T_v such that $\phi(w) \geq \phi(r)$ then we set the entry in row $\phi(v')$ of the new column to $\phi(w)$. Otherwise we leave the entry empty. We note the edge (v, r) that corresponds to each column.

$M(T)$ is a $k \times |\mathcal{N}^-(r)|$ matrix (where k is the number of levels) consisting of partially populated columns of consecutive non-increasing entries. Entries containing 0 are considered 'unpopulated'. The rows with populated entries lie in the range 1 to $\phi(r)-1$. Each row represents the restrictions imposed on the level planarity of T by directed subtrees whose roots lie on the corresponding level. Each column represents the restrictions imposed on the level planarity of T by the corresponding T_v. From here on, by referring to, say, the first entry in a column, we mean the first populated entry. We use `first`$(M(T), j)$ and `last`$(M(T), j)$ to denote the index of the first and last entries in column j respectively.

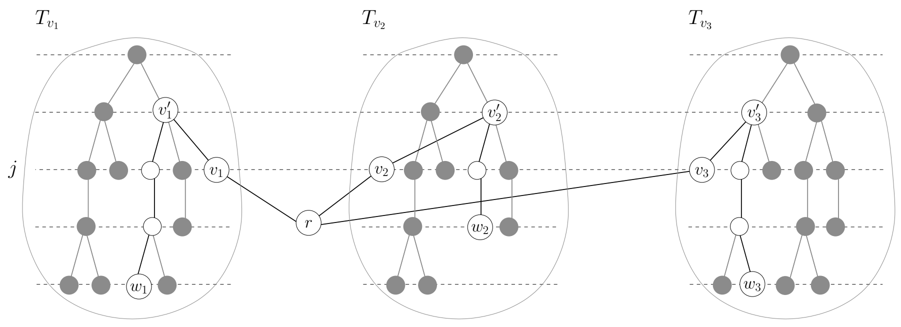


Figure 5.3: If row j of $M(T)$ has more than two (populated) entries then T with leveling ϕ is not level planar.

In the following lemma we show that if there are at most two entries in any one row of $M(T)$, each T_v can 'hang' over one side of r.

Lemma 5.2.1. *If T is a tree DAG with leveling ϕ and has one vertex of in-degree greater than one then T is level planar if and only if $M(T)$ has at most two (populated) entries in any one row.*

Proof: Suppose row j of $M(T)$ has three entries. Let $\{v_i\} \subseteq \mathcal{N}^-(r)$ and T_{v_i}, $1 \leq i \leq 3$, such that T_{v_i} is a directed tree whose column in $M(T)$ contains an entry in row j. Let $v_i', w_i \in V(T_{v_i})$ such that $\phi(v_i') = j$ and $\phi(w_i) \geq \phi(r)$, $1 \leq i \leq 3$. For any total order $<_j$ of the vertices in $\phi^{-1}(j)$ there must be at least one crossing in any leveled drawing of T (see Fig. 5.3). However, the crossings are avoidable if there are at most two populated entries in row j.

□

To insert a dummy vertex along an edge between some $v \in \mathcal{N}^-(r)$ and r we decrement each entry in the corresponding column, shift the column up one row (adding extra rows to the top of $M(T)$ if necessary) and repeatedly remove any last entry in the column whose value is now less than $\phi(r)$. We call this operation `shiftUp`$(M(T), j)$ where j is the index of the column to shift.

We wish to apply `shiftUp`$(M(T), j)$ a small number of times so that each row contains at most two entries. Lemma 5.2.1 suggests the following heuristic. Find the last row with more than two entries. Fix two columns whose last entries occur the latest (j_{L} and j_{R}). If more than two columns meet this criterion, choose two of these whose first entries occur the latest. If more than two columns meet both criteria, then the choice between these columns is arbitrary. Apply $\texttt{shiftUp}(M(T), j)$ to all other columns that have an entry in this row. Then go to the previous row and repeat. Once all the rows have been processed then $M(T)$ is left with at most two (populated) entries in any one row. Therefore inserting the new dummy vertices into T ensures level planarity. We will address the time complexity of this process in §5.2.4 and provide a complete worked example in §5.2.5. Before proceeding to the recursive case, we briefly consider the computational complexity of minimizing the number of dummy vertices for the base case.

Interpreting the Base Case as a Scheduling Problem

The problem of inserting a small number of dummy vertices to ensure the level planarity of T for the base case can be interpreted as a simple *task scheduling* problem. There are two identical processors, p_{L} and p_{R}, where p_{L} handles the tasks or columns assigned to j_{L} and likewise with p_{R} and j_{R}. There are at most $|\mathcal{N}^{-}(r)|$ independent tasks. The tasks are released in the (descending) order specified by $\texttt{last}(M(T), j)$ where j is the index of the corresponding column. The processing times for each task are the number of entries in the corresponding column but each task has the added peculiarity that making it wait (*i.e.* inserting a dummy vertex) may result in the task becoming temporarily unavailable and requiring a decreased processing time. Our goal is to minimize the sum of the waiting times for all available tasks so we are employing a shortest task first algorithm. This is optimal if the value of each entry is sufficiently large so that the peculiarity does not arise.

This interpretation shows the difficulty in minimizing the number of dummy vertices even for the base case. The peculiarity we referred to involves tasks with time dependent processing times: the length of the task depends on the time at which it is started. In classical task scheduling theory, task processing times are constant; however, there is a growing interest in scheduling models with time-dependent processing times (see Alidaee and Womer (1999) for a survey). We are particularly interested in a result of Cheng and Ding (1998); they prove that minimizing the total length of a schedule (the 'makespan') for a set of tasks on one processor with arbitrary release times and whose processing times decrease linearly at individual rates with respect to their starting times is strongly NP-Hard. The problem can be stated more formally as follows:

Problem 5.2.2 (MIN-MAKESPAN). *We are given a set of n independent tasks $\{T_j\}$. Each task T_j has a release time r_j, an initial processing time α_j and a decreasing processing rate w_j. r_j, α_j and w_j are all rational numbers. If a task T_j is scheduled to start at time $s_j \geq r_j$, then its actual processing time p_j is $\alpha_j - w_j s_j \geq 0$. We wish to find a schedule on one processor that minimizes the makespan, $\sum_{i=1}^{n} p_i$.*

Cheng and Ding (1998) proved that:

Theorem 5.2.3. *(Cheng and Ding, 1998)*
MIN-MAKESPAN *is strongly* NP*-Hard.*

We can formally state our problem as:

Problem 5.2.4 (MIN-DUMMY-VERTICES). *Given a tree DAG T with leveling ϕ and exactly one vertex of in-degree greater than one, find a minimum number of dummy vertices to ensure level planarity.*

This brings us to our result:

Theorem 5.2.5. MIN-DUMMY-VERTICES *is* NP*-Hard.*

Proof: We reduce MIN-MAKESPAN to MIN-DUMMY-VERTICES. The parameters in MIN-MAKESPAN may be rational so we first scale them up to integers. Then, for each task T_j, we add a group of directed trees to a tree DAG T. Every possible start time $s_j \geq r_j$ such that the actual processing time $\alpha_j - w_j s_j > 0$ contributes a single directed tree to this group.

More specifically, we create a tree DAG T with one vertex r of in-degree greater than one. T has a leveling ϕ with both *macro-* and *micro-levels*. The macro-levels are numbered 1 to $M = \max_j\{\alpha_j - w_j r_j\} + 2$. There are $\mu = \max_j\{\alpha_j - w_j r_j\}$ micro-levels in between each pair of consecutive macro-levels. We refer to a macro-level as being the p^{th} macro-level and to a micro-level as being the q^{th} micro-level above or below the p^{th} macro-level, $\forall 1 \leq q \leq \mu$ and $1 \leq p \leq M$. We place r on the $M - 1^{\text{th}}$ macro-level.

For each task T_j, we add a group of directed trees to T and join each one to r as follows. We add a directed tree for every possible start time $s_j \geq r_j$ such that the actual processing time $\alpha_j - w_j s_j > 0$. We place the root of this directed tree on the $M - (\alpha_j - w_j s_j) - s_j - 1^{\text{th}}$ macro-level, extend a branch down to a vertex on the $M - s_j - 1^{\text{th}}$ macro-level (this is the degree three vertex in each directed tree), then extend one branch from here to join r and another down to a vertex on the $(\alpha_j - w_j s_j)$th micro-level below the $M - 1^{\text{th}}$ macro-level. Figure 5.4 exemplifies this process for one task which results in the addition of three directed trees to T.

We repeat this process for each task. Figure 5.5 shows three groups of directed trees representing three tasks; the first having two directed trees, the second having three and the third having just one.

Finally, when the groups of directed trees for each task have been added, we add one last directed tree. We place the root of this tree on the 1^{st} macro-level, extend a branch to a vertex on the $M-2^{\text{th}}$ macro-level, then extend one branch from here to join r and another down to a vertex on the M^{th} macro-level. In effect, this directed tree keeps one of the two 'processors' (sides of r), p_{L} or p_{R} (j_{L} or j_{R}), occupied throughout. This is required since MIN-MAKESPAN is formulated in terms of one processor.

Within each group of directed trees for some task T_j, at most one directed tree can 'hang' on the free side of r. All others will be pushed up by the insertion of dummy vertices. The macro-level of the only degree-3 vertex in this directed tree, $M - s_j -$, determines the scheduled starting time s_j of the task T_j. This directed tree then occupies the free side of r for a number of

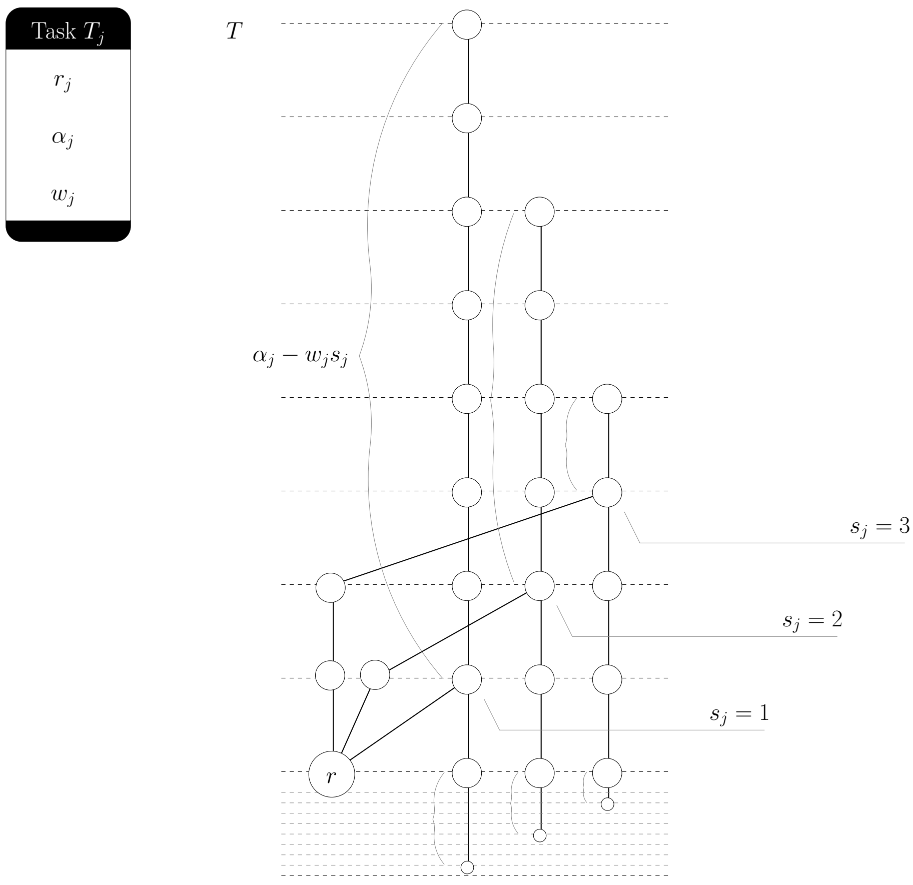


Figure 5.4: Every possible start time $s_j \geq r_j$ for a task T_j such that the actual processing time is greater than zero contributes a single directed tree to T.

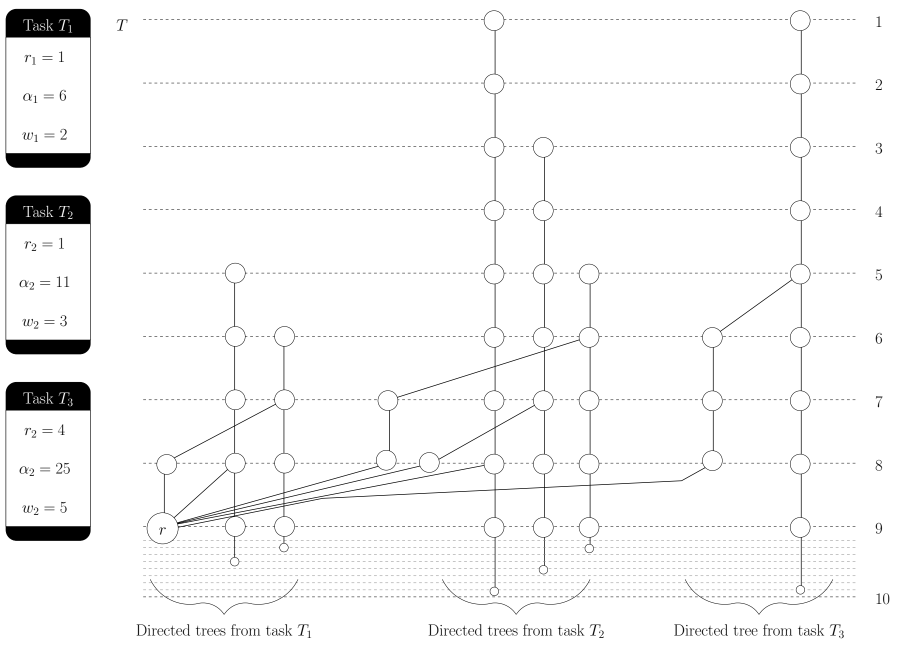


Figure 5.5: The three tasks T_1, T_2 and T_3 contribute three groups of directed trees to T.

macro-levels equivalent to the duration of its processing time, $\alpha_j - w_j s_j$. If all the directed trees in the group are pushed up then the task T_j is never started and its processing time is left decay to zero.

The final uppermost occupied macro-level of T determines the makespan of the schedule. If we minimize the number of dummy vertices then we are keeping this level as low as possible. The number of vertices in each T_j is bounded by a polynomial in the parameters r_j, a_j and w_j. Therefore, the total number of vertices in T is bounded by a polynomial in the parameters of MIN-MAKESPAN and, since MIN-MAKESPAN is strongly NP-Hard, our theorem follows.

$\square$

5.2.2 The Recursive Case

We now return to the recursive case of algorithm `makeTreeDAGLevelPlanar`. If $|\mathcal{V}| > 1$ then we choose some $r \in \mathcal{V}$ that maximizes $\phi(r)$. Note that the order in which we choose each r can be pre-computed. We proceed as before by populating a matrix $M(T)$. However, if any T_v is not a directed tree, we need to recursively ensure its level planarity. Let $M(T_v)$ be the final matrix associated with T_v. We populate a new column j_{T_v} of the present matrix $M(T)$ in two steps. Firstly, we visit each vertex v' along the (undirected) path from v to its earliest ancestor whose in-degree is anything other than one and set the entries as in the base case. Secondly, we populate column j_{T_v} of $M(T)$ bottom-up by setting $M(T)_{i,j_{T_v}}$ to

$$\begin{cases} \max\{maxRow, maxCol\} & \text{if } M(T)_{i,j_{T_v}} > 0 \wedge maxRow > 0 \\ maxCol & \text{if } M(T)_{i,j_{T_v}} > 0 \wedge maxRow = 0 \\ 0 & \text{if } M(T)_{i,j_{T_v}} = 0 \wedge maxRow = 0 \end{cases} \tag{5.1}$$

where $maxRow = \max_j M(T_v)_{i,j}$ and $maxCol = \max_{i'} M(T)_{i',j_{T_v}}$. This new column can be considered a substitute that is at least as restrictive as T_v

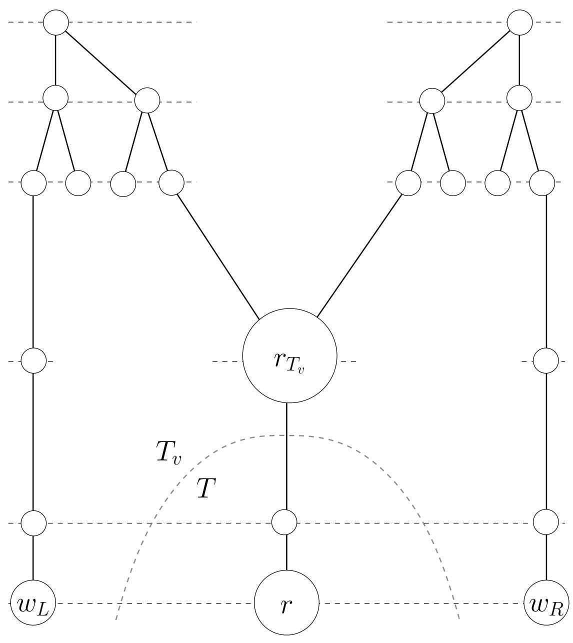


Figure 5.6: T_v is smothering r.

on the level planarity of T.

An additional complication occurs if some T_v *smothers* r (see Fig. 5.6 for a typical example). In this case T_v, whose level planarity we have already ensured, has three vertices $r_{T_v}, w_{\mathrm{L}}, w_{\mathrm{R}} \in V(T_v)$ such that r_{T_v} has in-degree greater than one, $\phi(w_{\mathrm{L}}), \phi(w_{\mathrm{R}}) \geq \phi(r)$ and the least common ancestors of the pairs w_{L}, r_{T_v} and r_{T_v}, w_{R} are on the same level. In other words, in any level planar drawing of T_v the undirected paths from r_{T_v} to w_{L} and from r_{T_v} to w_{R} 'hang' on either side of r.

In terms of $M(T)$, T_v smothers r if $M(T)$ has more than one entry in any one row whose index is less than $\phi(r_{T_v})$ and whose value is greater than or equal to $\phi(r)$. The solution is much the same as in the base case where we were required to shift the columns so that each row was left with at most two entries. We start with the last row with more than one entry whose index is less than $\phi(r_{T_v})$ and whose value is greater than or equal to $\phi(r)$. Using the same criteria as before, we fix one column (j_{F}) and apply `shiftUp`$(M(T_v), j)$ to the others. Then go to the previous row and repeat.

```
   Input: $T, \phi$
   Output: $M(T)$
1  $M(T) \leftarrow$ empty;
2  $col \leftarrow 1$;
3  Choose $r \in \{v \in V(T) :$ inDeg$(v) > 1\}$ that maximizes $\phi(r)$;
4  foreach $v \in \mathcal{N}^-(r)$ do
5      while $v \neq$ empty do
6          $entry \leftarrow \phi(v) +$ height(the tree in $T_v$ rooted at $v$);
7          if $entry \geq \phi(r)$ then
8              $M(T)_{\phi(v),col} \leftarrow entry$;
9          $v \leftarrow$ parent$(v)$;                /* empty if inDeg$(v) \neq 1$ */
10     if $T_v$ is not a directed tree then
11         $M(T_v) \leftarrow$ makeTreeDAGLevelPlanar$(T_v, \phi)$;
12         Use shortest task first heuristic when applying
           shiftUp$(M(T_v), j)$ to leave at most one (populated) entry in
           certain rows of $M(T_v)$ (see §5.2.2);
13     Combine $M(T_v)$ into column $col$ of $M(T)$;
14     $col \leftarrow col + 1$;
15 Use shortest task first heuristic when applying shiftUp$(M(T), j)$ to
   leave at most two (populated) entries in any one row of $M(T)$ (see
   §5.2.1);
16 return $M(T)$;
```

Algorithm 3: makeTreeDAGLevelPlanar

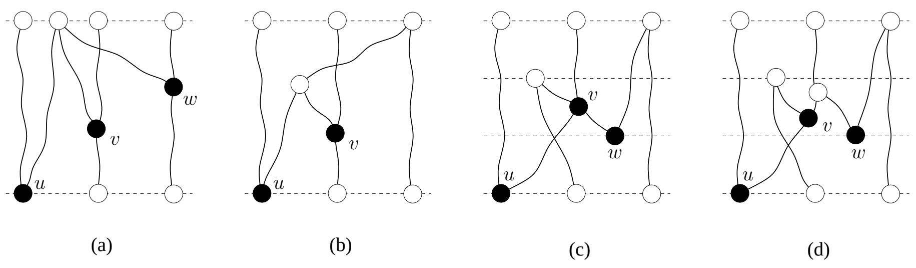


Figure 5.7: The four MLNP tree patterns: (a) P_1 (b) P_2 (c) P_3 (d) P_4.

5.2.3 Proof of Correctness

In this section we prove the correctness of our algorithm, `makeTreeDAGLevel-Planar`. Before this we introduce the concept of *level non-planar patterns*.

A *pattern* is a set of graphs that are homeomorphically equivalent. Any graph matching a *level non-planar pattern* (LNP) is itself level non-planar. Removing any edge from a graph that matches a *minimum level non-planar pattern* (MLNP) results in a graph that is level planar. Di Battista and Nardelli (1988) provided three MLNPs for hierarchies and proved their necessity and sufficiency. Healy et al. (2004) refined these to cover all leveled graphs but their characterization was shown to be incomplete. Fowler and Kobourov (2008) have completed the characterization for leveled trees. The four MLNP patterns for trees, P_1, P_2, P_3 and P_4, which are formally described in the work of Fowler and Kobourov (2008) are shown in Fig. 5.7. A tree T with leveling ϕ is level non-planar if and only if T has a subtree matching one of the MLNPs P_1, P_2, P_3, or P_4. We use this result as the basis for our proof of correctness.

Theorem 5.2.6. *Given a tree DAG T with leveling ϕ, `makeTreeDAGLevel-Planar`(T, ϕ) inserts dummy vertices to ensure level planarity.*

Proof: We show that every subtree of T matching one of the MLNPs P_1, P_2, P_3, or P_4 will have dummy vertices inserted so that it no longer matches any MLNP. P_1 has three vertices of in-degree greater than one. `makeTreeDAG-LevelPlanar` will recurse to the tree DAG whose root has the lowest $\phi(r)$. This tree DAG is level planar and will be combined into a single column and added to the tree DAG whose root has the next lowest $\phi(r)$. This causes

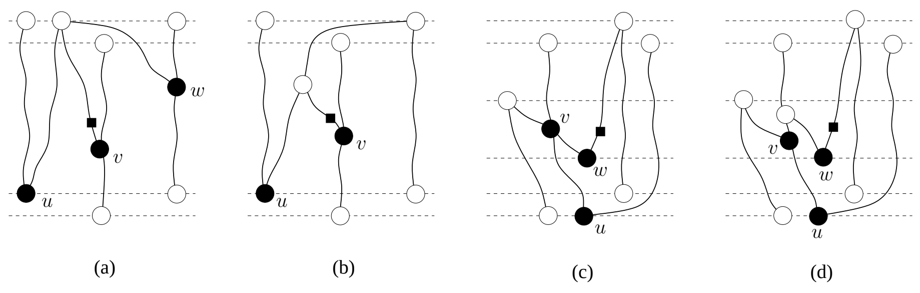


Figure 5.8: The four MLNP tree patterns with dummy vertices inserted: (a) P_1 (b) P_2 (c) P_3 (d) P_4.

smothering and requires the insertion of dummy vertices (see Fig. 5.8(a)) to ensure level planarity. Finally, this will be combined into a single column and added to the tree DAG whose root has the next lowest $\phi(r)$. We have now traversed the entire pattern P_1 and the insertion of the dummy vertices has made it level planar. Analogous arguments handle subtrees of T that match P_2, P_3, or P_4 (see Fig. 5.8(b) - (d)).

□

5.2.4 Implementation

By storing $M(T)$ in a *sparse form*, `makeTreeDAGLevelPlanar` can be modified to run in $\mathcal{O}(|V(T)|)$ time. For each column, we list an *edge identifier*, an *offset*, the index of the first row where the column has a populated entry, the number of *pairs* of entries and the entries themselves (see Fig. 5.9). The edge identifier is the edge into which we insert dummy vertices when applying `shiftUp`$(M(T), j)$. The offset is used by `shiftUp`$(M(T), j)$ to update all entries in the column in constant time. When a column is shifted up the offset is first decremented. The entries are encoded as pairs of values where the first is the number of times the second value appears consecutively, *e.g.* the entries $8, 7, 7, 5, 5$ are encoded as $1, 8, 2, 7, 2, 5$. If a column with those values is shifted up for the first time we set its offset to -1 and, if the second value in the last pair plus the offset is less than $\phi(r)$, we remove the last pair from the entries and decrement the number of distinct entries. The sparse form allows the `first`$(M(T), j)$, `last`$(M(T), j)$ and `shiftUp`$(M(T), j)$ operations to be

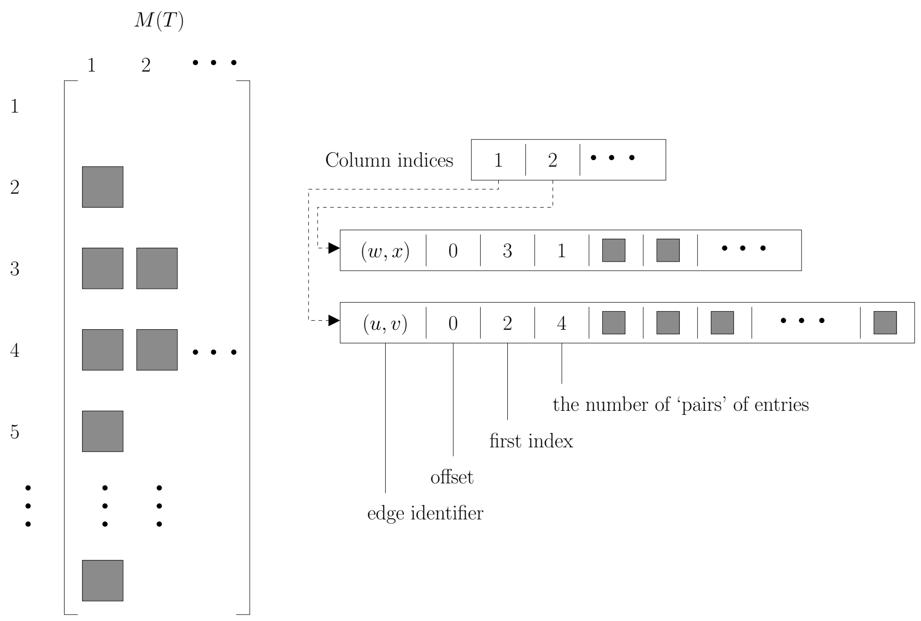


Figure 5.9: $M(T)$ can be stored in a sparse form.

performed in $\mathcal{O}(1)$ time.

The shortest task first heuristics (Lines **12** and **15** of Algorithm 3) can easily be implemented using two nested `for` loops. However, we need only visit the populated entries in each matrix. Using the sparse form of $M(T)$ in Line **12** we can sort the columns in descending order according to `last`$(M(T_v), j)$ using bucket sort (since the range of values is the range of levels occupied by the graph and must be less than or equal to the number of vertices) and then process each bucket from left to right as follows. Choose one of the columns with the greatest `first`$(M(T_v), j)$ to be j_{F}, apply `shiftUp`$(M(T_v), j)$ to the other columns and add them to the next bucket. A similar approach can be used in Line **15**.

5.2.5 A Worked Example

In this section we illustrate the workings of `makeTreeDAGLevelPlanar` with an example. Consider the tree DAG T in Fig. 5.10. It has two vertices of in-degree greater than one, namely r_1 and r_2. The algorithm visits r_1 but finds that one of its neighbors in $\mathcal{N}^-(r_1)$ is a tree DAG and needs to recursively ensure its level planarity first. After traversing the (undirected) paths from

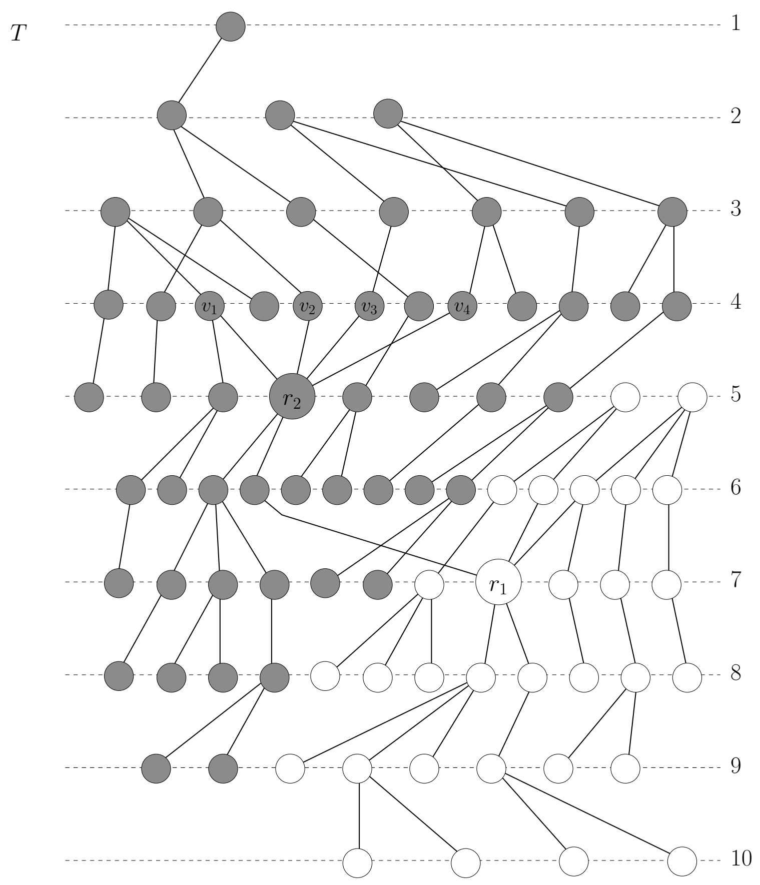


Figure 5.10: T has two vertices with in-degree greater than one, namely r_1 and r_2. We focus on the shaded sub-tree DAG rooted at r_1 first.

each $v \in \mathcal{N}^-(r_2)$ to the roots of their respective trees, we get a matrix for the sub-tree DAG T' (the graph induced by the shaded vertices in Fig. 5.10):

$$M(T') = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \\ \end{array} \begin{array}{cccc} (v_1,r_2) & (v_2,r_2) & (v_3,r_2) & (v_4,r_2) \end{array} \hspace{-0pt} \left[\begin{array}{cccc} & 6 & & \\ & 6 & 6 & 7 \\ 7 & 5 & & \\ 7 & & & \\ & & & \\ \vdots & \vdots & \vdots & \vdots \end{array} \right].$$

By Lemma 5.2.1, T' is not level planar. We fix the second and third columns and perform $\texttt{shiftUp}(M(T'), 4)$ to get the following matrix:

$$M(T') = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \\ \end{array} \begin{array}{cccc} (v_1, r_2) & (v_2, r_2) & (v_3, r_2) & (v_4, r_2) \\ & 6 & & 6 \\ & 6 & 6 & \\ 7 & 5 & & \\ 7 & & & \\ & & & \\ \vdots & \vdots & \vdots & \vdots \end{array} .$$

This equates to inserting a dummy vertex along the edge (v_4, r_2). T' is now level planar and we can turn to the level planarity of the entire tree T (see Fig. 5.11). After traversing the (undirected) paths from each $v \in \mathcal{N}^-(r_1)$ to the roots of their respective trees and combining $M(T')$ into the first column using Eqn. 5.1, we get a matrix for T:

$$M(T) = \begin{array}{c|ccc} & (v_1, r_1) & (v_2, r_1) & (v_3, r_1) \\ \hline 1 & 9 & & \\ 2 & 9 & & \\ 3 & 9 & & \\ 4 & 9 & & \\ 5 & 9 & 8 & 9 \\ 6 & & & 8 \\ & \vdots & \vdots & \vdots \end{array} .$$

By Lemma 5.2.1, T is not level planar. We fix the second and third columns and perform $\texttt{shiftUp}(M(T), 1)$ to get the following matrix:

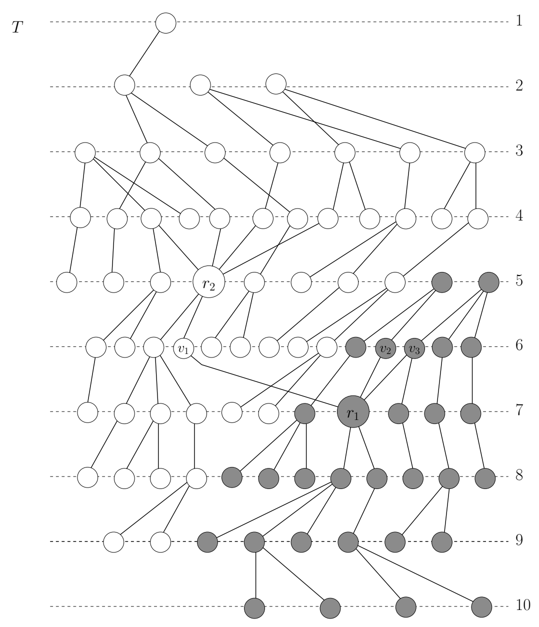


Figure 5.11: We focus on the sub-tree DAG (shaded) rooted at r_1 first.

$$M(T) = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ \\ \end{array} \begin{array}{ccc} (v_1, r_1) & (v_2, r_1) & (v_3, r_1) \\ \left[\begin{array}{c} 8 \\ 8 \\ 8 \\ 8 \\ 8 \\ \\ \\ \vdots \end{array}\right. & \begin{array}{c} \\ \\ \\ \\ \\ 8 \\ \\ \vdots \end{array} & \left.\begin{array}{c} \\ \\ \\ \\ \\ 9 \\ 8 \\ \vdots \end{array}\right] \end{array}.$$

This equates to inserting a dummy vertex along the edge (v_1, r_1). T is now level planar and we are done.

5.2.6 Violating the Leveling

While ensuring the level planarity of T, we may be violating the leveling with respect to G (see the edge (u, v) in Fig. 5.1(b)). These upwardly directed edges can be deceptive while trying to understand the graph visually. However, we

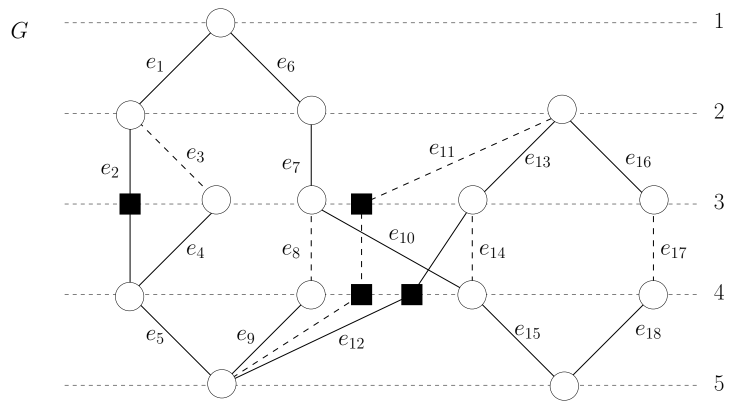


Figure 5.12: Every edge in $E(G) \setminus E(T)$ (the dashed edges) completes a fundamental cycle.

can insert compensating dummy vertices whenever the leveling with respect to G is violated.

We previously defined a leveling of G to be a surjective mapping $\phi : V(G) \to \{1, 2, \ldots, k\}$ where $\phi(v)$ specifies the level of each vertex v and, implicitly, the number of dummy vertices along each edge to make it proper. Alternatively, we can define a leveling of G to be a mapping $\psi : E(G) \to \mathbb{Z}^+$ where $\psi(e)$ specifies the number of dummy vertices along each edge e to make it proper and, implicitly, the level of each vertex.

At the same time, every edge in $E(G) \setminus E(T)$ completes an (undirected) *fundamental cycle* C in T (see Fig. 5.12). By traversing each C in some arbitrary direction, we get the *cycle vector* $\chi(C)$ (with coordinates $\chi(C)_e, \forall e \in E(G)$) defined by

$$\chi(C)_e = \begin{cases} 1 & \text{if e is directed with the traversal of C,} \\ -1 & \text{if e is directed against the traversal of C,} \\ 0 & \text{if e is not in C.} \end{cases} \tag{5.2}$$

The set of vectors corresponding to a set of fundamental cycles $\mathcal{C}$ constitute a basis for the cycle vector space of G (see Diestel (2005) for a more detailed treatment).

We say that a cycle is *balanced* with respect to some leveling ψ if

$$\sum_{e \in E(G)} \chi(C)_e(1 + \psi(e)) = 0. \tag{5.3}$$

We claim that if ψ balances a set of fundamental cycles then it corresponds to a leveling of G. We establish this fact with the following lemma and theorem:

Lemma 5.2.7. *If ψ balances a set of fundamental cycles in G then it balances all the cycles in G.*

Proof: Let $\mathcal{F} = \{F_1, \ldots, F_p\}$ be a set of balanced fundamental cycles in G. Therefore, $\sum_{e \in E(G)} \chi(F_i)_e(1 + \psi(e)) = 0, \forall i \in \{1, \ldots, p\}$.

Let C be any cycle in G. $\chi(C)$ must be a linear combination of the cycle vectors of the cycles in $\mathcal{F}$: $\chi(C) = \sum_{i \in \{1,\ldots,p\}} \alpha_i \chi(F_i)$ with at least one α_i being non-zero. So,

$$\begin{aligned}
\sum_{e \in E(G)} \chi(C)_e(1 + \psi(e)) &= \sum_{e \in E(G)} \sum_{i \in \{1,\ldots,p\}} \alpha_i \chi(F_i)_e(1 + \psi(e)) \\
&= \alpha_i \sum_{i \in \{1,\ldots,p\}} \sum_{e \in E(G)} \chi(F_i)_e(1 + \psi(e)) \\
&= \alpha_i \sum_{i \in \{1,\ldots,p\}} (0) = 0.
\end{aligned}$$

□

We now state and prove the theorem:

Theorem 5.2.8. *If ψ balances a set of fundamental cycles in G then it corresponds to a leveling of G.*

Proof: We find the corresponding ϕ by performing an (undirected) DFS on G. We assign a level to each newly visited vertex v as follows:

$$\phi(v) = \begin{cases} \phi(u) + \psi((u,v)) + 1 & \text{if } (u,v) \in E(G) \\ \phi(u) - \psi((v,u)) - 1 & \text{if } (v,u) \in E(G) \end{cases} \tag{5.4}$$

where u is v's predecessor in the DFS. We can assign the first visited vertex to any level.

It is clear that $\phi(v) - \phi(u) > 0$ holds for all tree edges in the DFS traversal. However, it remains to be shown that it holds for the back edges. Each back edge closes a cycle C in G made up of otherwise tree edges. Let $\chi(C)$ be the cycle vector of C in the direction in which it was traversed during the DFS. Let $(v_0, v_1, \ldots, v_q)$ and $(e_0, e_1, \ldots, e_q)$ be the sequences of vertices and edges in the cycle respectively. Without loss of generality, we can assume that $e_q = (v_q, v_0)$ is the back edge, so we are required to show that $\phi(v_0) - \phi(v_q) > 0$.

From the DFS traversal and Lemma 5.2.7 we get:

$$\begin{aligned}
\phi(v_q) &= \phi(v_0) + \sum_{i \in \{0,\ldots,q-1\}} \chi(C)_{e_i}(\psi(e_i) + 1) \\
&= \phi(v_0) + \sum_{i \in \{0,\ldots,q\}} \chi(C)_{e_i}(\psi(e_i) + 1) - \chi(C)_{e_q}(\psi(e_q) + 1) \\
&= \phi(v_0) - \chi(C)_{e_q}(\psi(e_q) + 1).
\end{aligned}$$

Since we assumed e_q to be the back edge, so $\chi(C)_{e_q} = 1$ and we get $\phi(v_0) - \phi(v_q) > 0$. To satisfy $\phi(u) > 0, \forall u \in V(G)$ we can 'shift' the leveling by adding $|\min_{v \in V(G)} \phi(v)| + 1$ to all $\phi(v)$.

$\square$

Now, suppose we partition the edge set $E(G) = \bigcup S = s_1 \cup \ldots \cup s_t$ such that each subset s_j, $1 \leq j \leq t$, is the maximal set of edges shared between the same subset of cycles in $\mathcal{C}$ (see Fig. 5.13). In other words, two edges belong to the same subset if they are both bridges or both belong to the same S-vertex of an SPQR-tree (Di Battista and Tamassia, 1996) of the same biconnected of the underlying undirected graph of G. Then, if ψ is some leveling of G, any dummy vertex along an edge $e \in s_j$, $1 \leq j \leq t$, can be moved to any other similarly directed edge (with respect to any cycle) in s_j. In fact, we can specify an equivalence class of levelings of G using two mappings $\psi_+ : S \to \mathbb{Z}^+$ and $\psi_- : S \to \mathbb{Z}^+$ where ψ_+ and ψ_- specify the number of dummy vertices

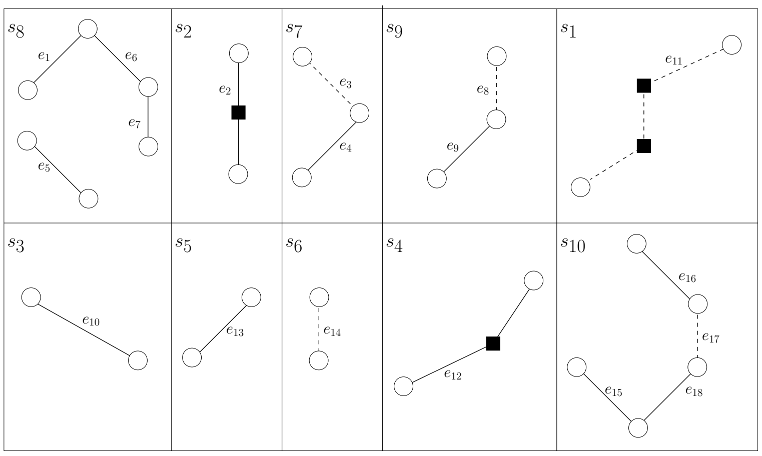


Figure 5.13: The partitioning of the edge set in Fig. 5.12.

along the edges in either direction in each subset s_j. To make sure that the directions are consistent, we use a *fundamental cycle-edge subset incidence matrix* $\mathcal{F}$ defined by

$$\mathcal{F}_{C,s} = \begin{cases} 1 & \text{if the directions of the edges in s} \\ & \text{are consistent with the traversal of C,} \\ -1 & \text{if the directions of the edges in s} \\ & \text{are inconsistent with the traversal of C,} \\ 0 & \text{if s is not in C.} \end{cases} \tag{5.5}$$

For example, a fundamental cycle-edge subset incidence matrix for the DAG in Fig. 5.12 is

$$\mathcal{F} = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \end{array} \begin{array}{c} \begin{array}{cccccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} \end{array} \\ \left[\begin{array}{cccccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right] \end{array}.$$

We define the vectors $\vec{w}$ (with coordinates $\vec{w}_s, \forall s \in S$, in the order of the columns of $\mathcal{F}$) by $\vec{w}_s = \psi_+(s) - \psi_-(s)$ and $\vec{b}$ (with coordinates $\vec{b}_C, \forall C \in \mathcal{C}$, in the order of the rows of $\mathcal{F}$) by $\vec{b}_C = \sum \chi(C)$. Now, if ψ_+ and ψ_- specify a leveling of G then

$$\mathcal{F}\vec{w} = \vec{b}, \tag{5.6}$$

i.e. the dummy vertices balance the fundamental cycles. This formulation has been used by Harrigan and Healy (2005) to find a leveling of a DAG G with the minimum number of dummy vertices.

So if we start with an initial leveling of G and add dummy vertices to ensure level planarity with respect to T, we can prevent any violation of the leveling by examining the null-space of $\mathcal{F}$. For example, if the edge e_{13} in s_5 needs an additional dummy vertex, this can be compensated by adding a dummy vertex to e_{11} in s_1 and a dummy vertex to one of e_{16}, e_{17} or e_{18} in s_{10} since the columns corresponding to s_1, s_5 and s_{10} in $\mathcal{F}$ are linearly dependent.

5.3 Crossing Minimization

An embedding algorithm is implied by `makeTreeDAGLevelPlanar`. Fixing columns (j_L, j_R and j_F) determines the relative order of their corresponding vertices on their respective levels. However, we have some freedom when positioning the remaining vertices. We use this freedom to reduce the number of edge crossings involving edges that are not part of the significant spanning tree $E(T)$.

We use an order induced by a Fiedler vector $\vec{x}$ of $\mathcal{L}(G)$: if the relative order of two vertices u, v on the same level has not been decided and $\vec{x}_u < \vec{x}_v$ then we set u to the left of v. This heuristic has previously been used to determine exact x-coordinates when drawing a directed graph (Carmel et al., 2004) and for the 2-level crossing minimization problem (Newton et al., 2003). It is based on a result of Shahrokhi et al. (2001) that shows a strong relation between the *minimum linear arrangement (MLA) problem* of a bipartite graph, the minimum number of edge crossings and an algorithm for finding an approximate solution to the MLA problem (Juvan and Mohar, 1992): given an undirected graph G, the MLA problem is to determine a bijection $f : V(G) \rightarrow \{1 \ldots n\}$ such that $\sum_{(u,v) \in E(G)} |f(u) - f(v)|$ is minimized. Juvan and Mohar (1992) use a Fiedler vector $\vec{x}$ of $\mathcal{L}(G)$ to induce an order (if $\vec{x}_u < \vec{x}_v$ then $f(u) < f(v)$) which is unique up to the relative order of repeated eigenvector elements. Shahrokhi et al. (2001) show that in most cases, if we have an approximate solution to the MLA problem for a bipartite graph $G = (V_1 \cup V_2, E)$, we can place the vertices of V_1 and V_2 on two levels in the order obtained from f to obtain a good approximation for the 2-level crossing minimization problem. Empirical evidence (Newton et al., 2003) suggests that this heuristic efficiently provides good solutions for the 2-level case and Carmel et al. (2004) have used it with much success over an arbitrary number of levels. It is worth noting that this order is determined globally, *i.e.* for all vertices on all levels at the same time.

5.4 Bringing It All Together

We now outline how the approach from the preceding sections can be used to draw an entire directed graph. The dependency relationship between packages in large software systems tend to have a specific structure. Packages depend on base packages with common functionality that is shared with many other packages and on certain application-specific packages. Supposing this graph has a significant spanning tree DAG, *e.g.* the tree that propagates the most centrality, we can draw it so that this spanning tree DAG is emphasized.

Figure 5.14 depicts the largest component of the dependency graph for packages of the Python programming language[1] after removing any redundant dependencies by computing its transitive reduction. The significant spanning tree (solid edges) was computed as in §5.1 and the orders of the vertices on each level were determined by the embedding algorithm implied by `makeTreeDAG-LevelPlanar` along with the crossing minimization heuristic. The computation of the final x-coordinates was based on the work of Brandes and Köpf (2002)

[1] `http://www.python.org/`

and the less significant edges (dashed edges) were added in by hand. The technique of §5.2.6 has not been applied.

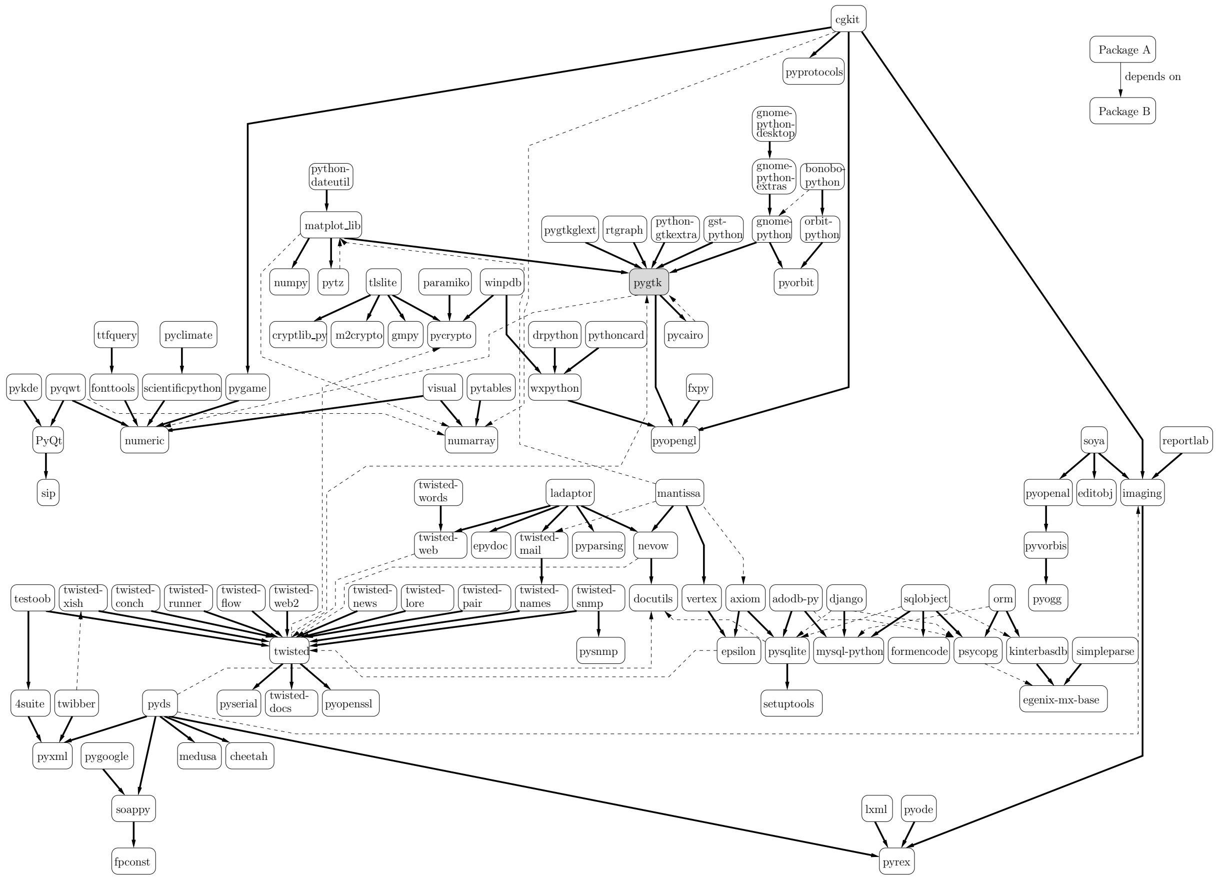


Figure 5.14: The dependency graph for Python packages.

Chapter 6

Conclusions and Open Problems

In summary, we have presented some new results regarding leveled drawings of graphs. We improved the level planarity testing and embedding algorithm of Healy and Kuusik (2004) to run entirely in quadratic running-time and to handle a family of embedding constraints. We reduced k-LEVEL-CROSSING-MIN to MAX-BALANCED-SUBGRAPH, thereby applying a variety of existing algorithms to the multi-level crossing minimization problem. For the special case of leveled trees, we investigated the complexity of the crossing minimization and maximum planarization problems for two levels, two levels with the order of the vertices on one of the levels fixed and for an arbitrary number of levels. We have also presented a method of drawing a leveled graph that emphasizes a significant spanning tree. This method ensures the level planarity of the spanning tree by inserting dummy vertices along edges to destroy any instances of the MLNP tree patterns. It relies on a new formulation of the leveling of a graph based on a 'balanced' set of fundamental cycles to prevent these newly inserted dummy vertices from violating the leveling.

Our results suggest a number of open problems. Kuusik (2000) conjectured that the minimum number of edge crossings in a leveled graph is less than or equal to the number of cycles having an odd number of '–'-labeled edges in any set of fundamental cycles of the vertex-exchange graph. While this conjecture remains unresolved, our reduction of k-LEVEL-CROSSING-

MIN to MAX-BALANCED-SUBGRAPH using the vertex-exchange graph establishes some other bounds on the minimum number of edge crossings in a leveled graph. In particular, the *frustration index* of a signed graph is the minimum number of edges whose deletion results in a balanced graph. Existing bounds on this index (Solé and Zaslavsky, 1994) must apply to the vertex-exchange graph and hence, to the minimum number of edge crossings in a leveled graph. Indeed, a future avenue of research is the application of other results from the theory of signed graphs (Zaslavsky, 1999) to the number of edge crossings in a leveled graph. For instance, given a maximum level planar subgraph $G = (V, E')$ of a leveled graph $G = (V, E)$, one may ask: is it possible to calculate a leveled drawing of G where the edges of E' do not cross each other and the number of edge crossings (between the edges in $E \setminus E'$, or between the edges in E and $E \setminus E'$) is a minimum? Results regarding the *restricted frustration index* of a signed graph could prove useful (Solé and Zaslavsky, 1994).

In studying the crossing minimization and maximum planarization problems with regard to k-leveled trees, we proved the boxed results in Tables 4.1 and 4.2. However, we would like to know the complexity of the maximum planarization problem for leveled trees with an arbitrary number of levels. A construction similar to the one used in the proof of Theorem 4.2.4 does not appear to work since edges in G' that are not part of some maximum level planar subgraph can extend all the way to the extreme left or right sides of the drawing taking their struts with them. Another follow-up question is, for what minimum values of k do the crossing minimization problem (and perhaps the maximum planarization problem) for k-leveled trees become NP-Hard? We would also like to extend the planarizing penalty digraph to handle graphs, and not just trees, on two levels where the order of the vertices on one of the levels is fixed.

We showed that by inserting dummy vertices along certain edges, we can ensure the level planarity of a significant spanning tree or k-leveled tree. These dummy vertices destroy the MLNP tree patterns (see the proof of Theorem 5.2.6). Can this approach be extended to the other MLNP patterns (Healy et al., 2004; Fowler and Kobourov, 2008) allowing us to ensure the level planarity of a wider class of graph? Such a method could be used to reassign vertices of a leveled graph to different levels in order to reduce the total number of edge crossings. We also believe that the methods of Chapter 3 could be developed into an improved heuristic over the one employed in §5.3. In particular, the question posed above with regard to a given maximum level planar subgraph and the restricted frustration index could be restated where $G = (V, E')$ is simply a significant subgraph that must be kept level planar. We then seek to minimize the number of edge crossings involving the remaining edges.

Bibliography

E. Albacea. A Linear Algorithm for Bipartite Drawing with Minimum Edge Crossings of Complete Binary Trees. *Philippine Computing Journal*, 1(1): 1–5, 2006.

B. Alidaee and N. Womer. Scheduling with Time Dependent Processing Times: Review and Extensions. *Journal of the Operational Research Society*, 50(7): 711–720, 1999.

R. Andreev, P. Healy, and N. Nikolov. Applying Ant Colony Optimization Metaheuristic to the DAG Layering Problem. In *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS'07)*, pages 1–9. IEEE International, 2007.

F. Aoudja, M. Laborie, and A. Saint-Paul. CASE: Automatic Generation of Electrical Diagrams. *Computer-Aided Design*, 18(7):356–360, 1986.

C. Bachmaier. *Circle Planarity of Level Graphs.* PhD thesis, University of Passau, 2004.

C. Bachmaier, W. Brunner, and C. König. Cyclic Level Planarity Testing and Embedding. In S. Hong, T. Nishizeki, and W. Quan, editors, *Proceedings of the 15th International Symposium on Graph Drawing (GD'07)*, pages 50–61. Springer, 2008.

F. Barahona and A. Mahjoub. Facets of the Balanced (Acyclic) Induced Subgraph Polytope. *Mathematical Programming*, 45(1–3):21–33, 1988.

O. Bastert and C. Matuszewski. Layered Drawings of Digraphs. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs, Methods and Models*, volume 2025, pages 87–120. Springer, 1999.

C. Batini, E. Nardelli, and R. Tamassia. A Layout Algorithm for Data Flow Diagrams. *IEEE Transactions on Software Engineering*, 12(4):538–546, 1986.

B. Berger and P. Shor. Approximation Algorithms for the Maximum Acyclic Subgraph Problem. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 236–243, 1990.

P. Bonacich. Factoring and Weighting Approaches to Status Scores and Clique Identification. *Journal of Mathematical Sociology*, 2:113–120, 1972.

K. Booth and G. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity using PQ-tree Algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

R. Botafogo, E. Rivlin, and B. Schneiderman. Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics. *ACM Transactions on Information Systems*, 10(2):142–180, 1992.

F. Brandenburg, D. Eppstein, M. Goodrich, S. Kobourov, G. Liotta, and P. Mutzel. Selected Open Problems in Graph Drawing. In G. Liotta, editor, *Proceedings of the 11th International Symposium on Graph Drawing (GD'03)*, pages 515–539. Springer, 2004.

U. Brandes. Drawing on Physical Analogies. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs, Methods and Models*, pages 71–86. Springer, 2001.

U. Brandes and S. Cornelsen. Visual Ranking of Link Structures. *Journal of Graph Algorithms and Applications (JGAA)*, 7(2):181–201, 2003.

U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations.* Springer, 2005.

U. Brandes and B. Köpf. Fast and Simple Horizontal Coordinate Assignment. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, pages 31–44. Springer, 2002.

U. Brandes, G. Shubina, R. Tamassia, and D. Wagner. Fast Layout Methods for Timetable Graphs. In J. Marks, editor, *Proceedings of the 8th International Symposium on Graph Drawing (GD'00)*, pages 289–309. Springer, 2001.

J. Branke, S. Leppert, M. Middendorf, and P. Eades. Width-Restricted Layering of Acyclic Digraphs with Consideration of Dummy Nodes. *Information Processing Letters*, 81(2):59–63, 2002.

C. Buchheim, M. Jünger, and S. Leipert. A Fast Layout Algorithm for k-Level Graphs. In J. Marks, editor, *Proceedings of the 8th International Symposium on Graph Drawing (GD'00)*, pages 229–240. Springer, 2001.

C. Buchheim, M. Jünger, and S. Leipert. Drawing Rooted Trees in Linear Time. *Software – Practice and Experience*, 36(6):651–665, 2006.

L. Carmel, D. Harel, and Y. Koren. Drawing Directed Graphs Using One-Dimensional Optimization. In S. Kobourov and M. Goodrich, editors, *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*, pages 193–206. Springer, 2002.

L. Carmel, D. Harel, and Y. Koren. Combining Hierarchy and Energy for Drawing Directed Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):46–57, 2004.

M. Carpano. Automatic Display of Hierarchized Graphs for Computer-Aided Decision Analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 10:705–715, 1980.

D. Cartwright and F. Harary. Structural Balance: A Generalization of Heider's Theory. *Psychological Review*, 63:277–293, 1956.

T. Catarci. The Assignment Heuristic for Crossing Reduction. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(3):515–521, 1995.

A. Chaudhary, D. Chen, X. Hu, M. Niemier, R. Ravichandran, and K. Whitton. Fabricatable Interconnect and Molecular QCA Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26 (11):1978–1991, 2007.

T. Cheng and Q. Ding. The Complexity of Scheduling Starting Time Dependent Tasks with Release Times. *Information Processings Letters*, 65(2): 75–79, 1998.

N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A Linear Algorithm for Embedding Planar Graphs using PQ-Trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985.

F. Chung. On Optimal Linear Arrangements of Trees. *Computers and Mathematics with Applications*, 10(1):43–60, 1984.

E. Coffman and R. Graham. Optimal Scheduling for Two-Processor Systems. *Acta Informatica*, 1:200–213, 1972.

B. DasGupta, G. Enciso, E. Sontag, and Y. Zhang. Algorithmic and Complexity Results for Decompositions of Biological Networks into Monotone Subsystems. *Biosystems*, 90(1):161–178, 2007.

C. Demestrescu and I. Finocchi. Breaking Cycles for Minimizing Crossings. *Journal of Experimental Algorithmics*, 6(2), 2001.

C. Demetrescu and I. Finocchi. Combinatorial Algorithms for Feedback Problems in Directed Graphs. *Information Processing Letters*, 86(3):129–136, 2003.

G. Di Battista and E. Nardelli. Hierarchies and Planarity Theory. *IEEE Transactions on Systems, Man and Cybernetics*, 18(6):1035–1046, 1988.

G. Di Battista and R. Tamassia. On-Line Maintenance of Triconnected Components with SPQR-Trees. *Algorithmica*, 15(4):302–318, 1996.

G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics.* Springer, 3$^{\text{rd}}$ edition, 2005.

H. Do Nascimento and P. Eades. User Hints for Directed Graph Drawing. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, pages 205–219. Springer, 2001.

M. Doorley. *Automatic Levelling and Layout of Data Flow Diagrams.* PhD thesis, University of Limerick, 1995.

S. Dresbach. A New Heuristic Layout Algorithm for Directed Acyclic Graphs. In U. Derigs, A. Bachem, and A. Drexl, editors, *Operations Research Proceedings*, pages 121–126. Springer, 1994.

V. Dujmović and S. Whitesides. An Efficient Fixed Parameter Tractable Algorithm for 1-Sided Crossing Minimization. *Algorithmica*, 40(1):15–31, 2004.

V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, R. Ragde, F. Rosamond, M. Suderman, S. Whitesides, and D. Wood. A Fixed-Parameter Approach to 2-Layer Planarization. *Algorithmica*, 45(2):159–182, 2006.

V. Dujmović, H. Fernau, and M. Kaufmann. Fixed Parameter Algorithms for One-Sided Crossing Minimization – Revisited. *Journal of Discrete Algorithms*, to appear.

T. Dwyer and Y. Koren. Dig-CoLa: Directed Graph Layout through Constrained Energy Minimization. In *Proceedings of the IEEE Symposium on*

Information Visualization (InfoVis'05), pages 65–72. IEEE Computer Society, 2005.

T. Dwyer, Y. Koren, and K. Marriott. Drawing Directed Graphs using Quadratic Programming. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):536–548, 2006a.

T. Dwyer, Y. Koren, and K. Marriott. Stress Majorization with Orthogonal Ordering Constraints. In P. Healy and N. Nikolov, editors, *Proceedings of the 13th International Symposium on Graph Drawing (GD'05)*, pages 141–152. Springer, 2006b.

P. Eades. A Heuristic for Graph Drawing. *Congressus Numerantium*, 42: 149–160, 1984.

P. Eades and D. Kelly. Heuristics for Reducing Crossings in 2-Layered Networks. *Ars Combinatoria*, 21(A):89–98, 1986.

P. Eades and X. Lin. A New Heuristic for the Feedback Arc Set Problem. *Australian Journal of Combinatorics*, 12:15–26, 1995.

P. Eades and K. Sugiyama. How to Draw a Directed Graph. *Journal of Information Processing*, 13(4):424–437, 1991.

P. Eades and S. Whitesides. Drawing Graphs in Two Layers. *Theoretical Computer Science*, 131(2):361–374, 1994.

P. Eades and N. Wormald. Edge Crossings in Drawings of Bipartite Graphs. *Algorithmica*, 11(4):379–403, 1994.

P. Eades, X. Lin, and W. Smyth. A Fast and Effective Heuristic for the Feedback Arc Set Problem. *Information Processing Letters*, 47(6):319–323, 1993.

H. Eichelberger. *Aesthetics and Automatic Layout of UML Class Diagrams.* PhD thesis, University of Würzburg, 2005.

M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An Efficient Implementation of Sugiyama's Algorithm for Layered Graph Drawing. In J. Pach, editor, *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, pages 155–166. Springer, 2004.

T. Eschbach, W. Günther, R. Drechsler, and B. Becker. Crossing Reduction by Windows Optimization. In S. Kobourov and M. Goodrich, editors, *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*, pages 285–294. Springer, 2003.

L. Euler. Elementa Doctrinae Solidorum. *Novi Commentarii academiae scientiarum Petropolitanae*, 4:109–140, 1758.

H. Fernau. Two-Layer Planarization: Improving on Parameterized Algorithmics. *Journal of Graph Algorithms and Applications*, 9(2):205–238, 2005.

M. Flood. Exact and Heuristic Algorithms for the Weighted Feedback Arc Set Problem: A Special Case of the Skew-Symmetric Quadratic Assignment Problem. *Networks*, 20(1):1–23, 1990.

J. Fowler and S. Kobourov. Minimum Level Nonplanar Patterns for Trees. In S. Hong, T. Nishizeki, and W. Quan, editors, *Proceedings of the 15th International Symposium on Graph Drawing (GD'07)*, pages 69–75. Springer, 2008.

T. Fruchterman and E. Reingold. Graph Drawing by Force-directed Placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991.

E. Gansner, E. Koutsofios, S. North, and K. Vo. A Technique for Drawing Directed Graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

E. Gansner, Y. Koren, and S. North. Graph Drawing by Stress Majorization. In J. Pach, editor, *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, pages 239–250. Springer, 2004.

M. Garey and D. Johnson. Crossing Number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316, 1983.

M. Goemans and D. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems using Semidefinite Programming. *Journal of the ACM*, 42:1115–1145, 1995.

S. Goodman, S. Hedetniemi, and P. Slater. Advances on the Hamiltonian Completion Problem. *Journal of the ACM*, 22(3):352–360, 1975.

M. Grötschel, M. Jünger, and G. Reinelt. On the Acyclic Subgraph Polytope. *Mathematical Programming*, 33(1):28–42, 1985.

W. Günther, R. Schönfeld, B. Becker, and P. Molitor. k-Layer Straightline Crossing Minimization by Speeding Up Sifting. In J. Marks, editor, *Proceedings of the 8th International Symposium on Graph Drawing (GD'00)*, pages 253–258. Springer, 2001.

C. Gutwenger, K. Klein, and P. Mutzel. Planarity Testing and Optimal Edge Insertion with Embedding Constraints. In M. Kaufmann and D. Wagner, editors, *Proceedings of the 14th International Symposium on Graph Drawing (GD'06)*, pages 126–137. Springer, 2007.

F. Harary. On the Notion of Balance of a Signed Graph. *The Michigan Mathematical Journal*, 2(2):143–146, 1953–1954.

F. Harary and J. Kabell. A Simple Algorithm to Detect Balance in Signed Graphs. *Mathematical Social Sciences*, 1(1):131–136, 1980.

F. Harary and A. Schwenk. Trees with Hamiltonian Square. *Mathematiks*, 18: 138–140, 1971.

F. Harary and A. Schwenk. A New Crossing Number for Bipartite Graphs. *Utilitas Mathematica*, 1:203–209, 1972.

D. Harel and Y. Koren. A Fast Multi-Scale Method for Drawing Large Graphs. *Journal of Graph Algorithms and Applications*, 6(3):179–202, 2002.

M. Harrigan and P. Healy. On Layering Directed Acyclic Graphs. In M. Jünger, S. Kobourov, and P. Mutzel, editors, *Proceedings of Dagstuhl Seminar 05191*, volume 05191 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, 2005.

M. Harrigan and P. Healy. Efficiently Drawing a Significant Spanning Tree of a Directed Graph. In S. Hong and K. Ma, editors, *APVIS '07*, pages 35–42. IEEE Visualization and Graphics Technical Committee (VGTC), 2007.

M. Harrigan and P. Healy. Practical Level Planarity Testing and Layout with Embedding Constraints. In S. Hong, T. Nishizeki, and W. Quan, editors, *Proceedings of the 15th International Symposium on Graph Drawing (GD'07)*, pages 62–68. Springer, 2008a.

M. Harrigan and P. Healy. Using a Significant Spanning Tree to Draw a Directed Graph. *Journal of Graph Algorithms and Applications (to appear)*, 2008b.

P. Healy and A. Kuusik. Algorithms for Multi-Level Graph Planarity Testing and Layout. *Theoretical Computer Science*, 320(2–3):331–344, 2004.

P. Healy and N. Nikolov. A Branch-and-Cut Approach to the Directed Acyclic Graph Layering Problem. In S. Kobourov and M. Goodrich, editors, *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*, pages 98–109. Springer, 2002.

P. Healy, A. Kuusik, and S. Leipert. A Characterization of Level Planar Graphs. *Discrete Mathematics*, 280(1-3):51–63, 2004.

L. Heath and S. Pemmaraju. Recognizing Leveled-Planar DAGs in Linear Time. In F. Brandenburg, editor, *Proceedings of the 3rd International Symposium on Graph Drawing (GD'95)*, pages 300–311. Springer, 1996.

L. Heath and S. Pemmaraju. Stack and Queue Layouts of Directed Acyclic Graphs: Part II. *SIAM Journal on Computing*, 28(5):1588–1626, 1999.

H. Hotelling. Analysis of a Complex of Statistical Variables into Principal Components. *Journal of Educational Psychology*, 24:417–441, 1933.

F. Hüffner, N. Betzler, and R. Niedermeier. Optimal Edge Deletions for Signed Graph Balancing. In *WEA'07*, pages 297–310. Springer, 2007.

G. Jourdan, I. Rival, and N. Zaguia. Degree Navigator™: The Journey of a Visualization Software. In G. Liotta, editor, *Proceedings of the 11th International Symposium on Graph Drawing (GD'03)*, pages 491–493. Springer, 2004.

M. Jünger and S. Leipert. Level Planar Embedding in Linear Time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002.

M. Jünger and P. Mutzel, editors. *Graph Drawing Software.* Mathematics and Visualization. Springer, 2004.

M. Jünger and P. Mutzel. 2-Layer Straight Line Crossing Minimization: Performance of Exact and Heuristic Algorithms. *Journal of Graph Algorithms and Applications*, 1:1–25, 1997.

M. Jünger, E. Lee, P. Mutzel, and T. Odenthal. A Polyhedral Approach to the Multi-Layer Crossing Minimization Problem. In G. Di Battista, editor, *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*, pages 13–24. Springer, 1998.

M. Jünger, S. Leipert, and P. Mutzel. Level Planarity Testing in Linear Time. In S. Whitesides, editor, *Proceedings of the 6th International Symposium on Graph Drawing (GD'98)*, pages 224–237. Springer, 1999.

D. Jungnickel. *Graphs, Networks and Algorithms.* Algorithms and Computation in Mathematics. Springer, 2nd edition, 1999.

M. Juvan and B. Mohar. Optimal Linear Labelings and Eigenvalues of Graphs. *Discrete Applied Mathematics*, 36(2):153–168, 1992.

A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. Ontology Visualization Methods –A Survey. *ACM Computing Surveys*, 39 (4):1–43, 2007.

M. Kaufmann and D. Wagner, editors. *Drawing Graphs, Methods and Models.* Springer, 2001.

Y. Koren and D. Harel. A Multi-Scale Algorithm for the Linear Arrangement Problem. In L. Kucera, editor, *Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'02)*, pages 296–309. Springer, 2002.

Y. Koren, L. Carmel, and D. Harel. Drawing Huge Graphs by Algebraic Multigrid Optimization. *SIAM Journal on Multiscale Modeling and Simulation*, 1(4):645–673, 2003.

D. Koschützki, K. Lehmann, L. Peeters, S. Richter, D. Tenfelde-Podehl, and O. Zlotowski. Centrality Indices. In U. Brandes and T. Erlebach, editors, *Network Analysis*, pages 16–61. Springer, 2005.

E. Kruja, J. Marks, A. Blair, and R. Waters. A Short Note on the History of Graph Drawing. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, pages 272–286. Springer, 2002.

P. Kuntz, B. Pinaud, and R. Lehn. Elements for the Description of Fitness Landscapes Associated with Local Operators for Layered Drawings of Directed Graphs. In M. Resende, J. De Sousa, and A. Viana, editors, *Metaheuristics: Computer Decision-Making*, volume 86 of *Applied Optimization*, pages 405–420. Kluwer Academic Publishers, 2004.

P. Kuntz, B. Pinaud, and R. Lehn. Minimizing Crossings in Hierarchical Digraphs with a Hybridized Genetic Algorithm. *Journal of Heuristics*, 12 (1–2):23–36, 2006.

A. Kuusik. *Integer Linear Programming Approaches to Hierarchical Graph Drawing.* PhD thesis, University of Limerick, 2000.

M. Laguna, R. Martì, and V. Valls. Arc Crossing Minimization in Hierarchical Digraphs with Tabu Search. *Computers and Operations Research*, 24(12): 1175–1186, 1997.

K. Lehmann and S. Kottler. Visualizing Large and Clustered Networks. In M. Kaufmann and D. Wagner, editors, *Proceedings of the 14th International Symposium on Graph Drawing (GD'06)*, pages 240–251. Springer, 2007.

S. Leipert. *Level Planarity Testing and Embedding in Linear Time.* PhD thesis, University of Cologne, 1998.

T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley Sons Inc., 1990.

C. Lent and P. Tougaw. A Device Architecture for Computing with Quantum Dots. *Proceedings of the IEEE*, 85(4):541–557, 1997.

S. Lim and M. Niemier. Partitioning and Placement for Buildable QCA Circuits. In S. Shukla and R. Bahar, editors, *Nano, Quantum and Molecular Computing: Implications to High Level Design and Validation*, chapter Partitioning and Placement for Buildable QCA Circuits, pages 295–316. Kluwer Academic Publishers, 2004.

E. Loukakis. A Dynamic Programming Algorithm to Test a Signed Graph for Balance. *International Journal of Computer Mathematics*, 80(4):499–507, 2003.

E. Mäkinen. Experiments on Drawing 2-Level Hierarchical Graphs. *International Journal of Computer Mathematics*, 36:175–181, 1990.

E. Mäkinen and M. Sieranta. Genetic Algorithms for Drawing Bipartite Graphs. *International Journal of Computer Mathematics*, 53:157–166, 1994.

R. Martì and M. Laguna. Heuristics and Meta-Heuristics for 2-Layer Straight Line Crossing Minimization. *Discrete Applied Mathematics*, 127(3):665–678, 2003.

C. Matuszewski, R. Schönfeld, and P. Molitor. Using Sifting for k-Layer Straightline Crossing minimization. In Jan Kratochvíl, editor, *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, pages 217–224. Springer, 2000.

J. Mitchell and B. Borchers. Solving Real-World Linear Ordering Problems using a Primal-Dual Interior Point Cutting Plane Method. *Annals of Operations Research*, 62(1):253–276, 1996.

X. Muñoz, W. Unger, and I. Vrťo. One Sided Crossing Minimization is NP-Hard for Sparse Graphs. In S. Kobourov and M. Goodrich, editors, *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*, pages 115–123. Springer, 2003.

P. Mutzel. An Alternative Method to Crossing Minimization on Hierarchical Graphs. *SIAM Journal on Optimization*, 11(4):1065–1080, 2001a.

P. Mutzel. Optimization in Leveled Graphs. In C. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, chapter Optimization in Leveled Graphs, pages 189–196. Kluwer Academic Publishers, 2001b.

P. Mutzel. An Alternative Method to Crossing Minimization on Hierarchical Graphs. In S. North, editor, *Proceedings of the 4th International Symposium on Graph Drawing (GD'96)*, pages 318–333. Springer, 1997.

P. Mutzel and R. Weiskircher. Two-Layer Planarization in Graph Drawing. In K. Chwa and O. Ibarra, editors, *Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC'98)*, pages 69–78. Springer, 1998.

H. Nagamochi. An Improved Approximation to the One-Sided Bilayer Drawing. In G. Liotta, editor, *Proceedings of the 11th International Symposium on Graph Drawing (GD'03)*, pages 406–418. Springer, 2004.

H. Nagamochi. On the One-Sided Crossing Minimization in a Bipartite Graph with Large Degrees. *Theoretical Computer Science*, 322(1–3):417–446, 2005a.

H. Nagamochi. An Improved Bound on the One-Sided Minimum Crossing Number in Two-Layered Drawings. *Discrete Computational Geometry*, 33(4):569–591, 2005b.

F. Newbery. Edge Concentration: A Method for Clustering Directed Graphs. *ACM SIGSOFT Software Engineering Notes*, 14(7):76–85, 1989.

M. Newman and M. Girvan. Finding and Evaluating Community Structure in Networks. *Physical Review E*, 69(2):026113, 2004.

M. Newton, O. Sýkora, and I. Vrto. Two New Heuristics for Two-Sided Bipartite Graph Drawing. In S. Kobourov and M. Goodrich, editors, *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*, pages 312–319. Springer, 2003.

N. Nikolov. *A Polyhedral Approach to Directed Acyclic Graph Layering*. PhD thesis, University of Limerick, 2002.

N. Nikolov, A. Tarassov, and J. Branke. In Search for Efficient Heuristics for Minimum-Width Graph Layering with Consideration of Dummy Nodes. *ACM Journal of Experimental Algorithms*, 10(2.7), 2005.

J. Nummenmaa and J. Tuomi. Constructing Layouts for ER-Diagrams from Visibility-Representations. In H. Kangassalo, editor, *Proceedings of the 9th International Conference on Entity-Relationship Approach (ER'90)*, pages 315–329. ER Institute, 1990.

L. Protsko, P. Sorenson, J. Tremblay, and D. Schaefer. Towards the Automatic Generation of Software Diagrams. *IEEE Transactions on Software Engineering*, 17(1):10–21, 1991.

H. Purchase. Which Aesthetic has the Greatest Effect on Human Understanding? In G. Di Battista, editor, *Proceedings of the* th *International Symposium on Graph Drawing (GD'97)*, pages 248–261. Springer, 1998.

B. Randerath, E. Speckenmeyer, E. Boros, P. Hammer, A. Kogan, K. Makino, B. Simeone, and O. Cepek. A Satisfiability Formulation of Problems on Level Graphs. Technical report 40-2001, Rutgers Center for Operations Research, Rutgers University, June 2001.

R. Ravichandran, S. Lim, and M. Niemier. Automatic Cell Placement for Quantum-dot Cellular Automata. *Integration, the VLSI Journal*, 38(3): 541–548, 2005.

E. Reingold and J. Tilford. Tidier Drawings of Trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, 1981.

Y. Saab. A Fast and Effective Algorithm for the Feedback Arc Set Problem. *Journal of Heuristics*, 7(3):235–250, 2001.

G. Sander. Graph Layout through the VCG Tool. In R. Tamassia and I. Tollis, editors, *Proceedings of the* 2^{nd} *International Symposium on Graph Drawing (GD'94)*, pages 194–205. Springer, 1995.

G. Sander. Graph Layout for Applications in Compiler Construction. *Theoretical Computer Science*, 217(2):175–214, 1999.

M. Sarrafzadeh and C. Wong. *An Introduction to VLSI Physical Design*. McGraw-Hill College, 1996.

J. Seemann. Extending the Sugiyama Algorithm for Drawing UML Class Diagrams: Towards Automatic Layout of Object-Oriented Software Diagrams.

In G. Di Battista, editor, *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*, pages 415–424. Springer, 1997.

F. Shahrokhi and I. Vrťo. On 3-Layer Crossings and Pseudo Arrangements. In J. Kratochvíl, editor, *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, pages 225–231. Springer, 2000.

F. Shahrokhi, O. Sýkora, L. Székely, and I. Vrťo. On Bipartite Drawings and the Linear Arrangement Problem. *SIAM Journal on Computing*, 30(6): 1773–1789, 2001.

Y. Shiloach. A Minimum Linear Arrangement Algorithm for Undirected Trees. *SIAM Journal on Computing*, 8(3):422–430, 1979.

P. Solé and T. Zaslavsky. A Coding Approach to Signed Graphs. *SIAM Journal on Discrete Mathematics*, 7(4):544–553, 1994.

M. Suderman and S. Whitesides. Experiments with the Fixed-Parameter Approach for Two-Layer Planarization. *Journal of Graph Algorithms and Applications*, 9(1):149–163, 2005.

K. Sugiyama and K. Misue. Graph Drawing by the Magnetic Spring Model. *Journal of Visual Languages and Computing*, 6(3):217–231, 1995.

K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

K. Takamizawa, T. Nishizeki, and N. Saito. Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs. *Journal of the ACM*, 29 (3):623–641, 1982.

R. Tamassia. Constraints in Graph Drawing Algorithms. *Constraints*, 3(1): 87–120, 1998.

N. Tomii, Y. Kambayashi, and S. Yajima. On Planarization Algorithms of 2-Level Graphs. Technical Report EC77-38, Institute of Electronic Communications Engineers of Japan (IECEJ), 1977.

J. Utech, J. Branke, H. Schmeck, and P. Eades. An Evolutionary Algorithm for Drawing Directed Graphs. In *Proceedings of the International Conference on Imaging Science, Systems and Technology*, pages 154–160. CSREA Press, 1998.

V. Valls, R. Martì, and P. Lino. A Branch and Bound Algorithm for Minimizing the Number of Crossing Arcs in Bipartite Graphs. *European Journal of Operational Research*, 90:303–319, 1996.

J. Walker. A Node-Positioning Algorithm for General Trees. *Software – Practice and Experience*, 20(7):685–705, 1990.

J. Warfield. Crossing Theory and Hierarchy Mapping. *IEEE Transactions on Systems, Man and Cybernetics*, 7:505–523, 1977.

M. Waterman and J. Griggs. Interval Graphs and Maps of DNA. *Bulletin of Mathematical Biology*, 48(2):189–195, 1986.

X. Y-Li and M. Sallmann. New Bounds on the Barycenter Heuristic for Bipartite Graph Drawing. *Information Processing Letters*, 82:293–298, 2002.

A. Yamaguchi and A. Sugimoto. An Approximation Algorithm for the Two-Layered Graph Drawing Problem. In T. Asano, H. Imai, D. Lee, S. Nakano, and T. Tokuyama, editors, *COCOON '99*, pages 81–91. Springer, 1999.

T. Zaslavsky. Bibliography of Signed and Gain Graphs. *Electronic Journal of Combinatorics*, DS8, 1999.

L. Zheng and C. Buchheim. A New Exact Algorithm for the Two-Sided Crossing Minimization Problem. In A. Dress, Y. Xu, and B. Zhu, editors, *Proceedings of the 1st International Conference on Combinatorial Optimization and Application (COCOA'07)*, pages 301–310. Springer, 2007.

Index

Symbols

2-SAT . . . 33
k-LEVEL-CROSSING-MIN . . 53, 55, 56, 64, 101, 102
st-graph . . . 27
  numbering . . . 27

A

adjacency matrix . . . 13, 20–22, 24, 25
applications . . . 2
  DNA mapping . . . 2
  quantum-dot cellular automata 2
  VLSI layout . . . 2

B

balance . . . 17, 18, 28, 53, 102
balanced subgraph
  maximum . . . 8, 28, 53
bipartite . . . 16, 97, 98
bounded search tree . . . 54

C

caterpillar . . . 12, 68
  backbone . . . 12, 68
centrality
  eigenvector . . . 77, 98
clique . . . 11
  directed . . . 13, 69–72
complement . . . 11
component . 11, 29–35, 39–41, 63, 76, 98
  biconnected . . . 95
  dependent . . . 30, 33
  non-trivial . . . 11, 28, 30
  trivial . . . 11
constraint tree . . . 50, 52
constraint vertices
  grouping . . . 50
  mirroring . . . 50
  orientation . . . 50
cut . . . 18, 28
cut set . . . 11
cut vertex . . . 11, 64, 65
cycle 8, 11, 13, 16, 24, 32, 34, 40, 52, 74, 75, 93–95, 101
  balance . . . 101
  balanced . . . 93, 94, 97
  directed . . . 13, 63, 70, 71, 74, 75
  fundamental . . . 93, 94, 97, 101
  Hamiltonian . . . 11, 69
  vector . . . 93–95
  vector space . . . 93

D

digraph .... *see* directed graph
directed graph . 4, 6, 8–10, 12, 13, 62, 69–71, 74–77, 92, 97, 98
  acyclic 4, 13, 62–64
  underlying undirected .. 6, 12, 13, 95
  weighted 6
directed tree 12, 77, 79, 81–84
drawing 1, 4–6, 8, 15, 57
  k-leveled ..... *see* drawing,leveled
  leveled 2, 4, 5, 8, 9, 15, 16, 55, 74, 75, 79, 85, 102

E

edge 1, 10
  directed 1, 12
    head 12
    tail 12
  incident 10
  independent ... 10, 16, 18, 22, 25, 29, 34, 43, 53, 69
  long 27
  loop 10, 52
  multiple 10
  weighted 1, 11, 74, 77
embedding 27, 76
  k-leveled .. *see* embedding,leveled
  constraints . 8, 26, 27, 49, 50, 101
  cyclic leveled 49
  leveled . 15, 18, 20, 26–29, 32–34, 36, 37, 39, 50, 52, 53, 74

F

feedback arc set 62
  minimum 62
Fiedler vector .. 14, 59, 74–76, 97, 98
force-directed and energy minimization methods 5
  energy function 8
  stress function 8
forest 11, 12, 63, 68
  spanning 12
  subforest 12
frustration index 102
fundamental cycle-edge subset incidence matrix 96

G

graph . 1, 4, 10–14, 17, 18, 20, 21, 68, 69, 74, 87, 98
  k-connected 11
  acyclic 11
  biconnected 11, 52
  connected 11, 12, 76
  cyclic leveled 49
  disconnected 11
  leveled 2, 4, 5, 9, 14–17, 20, 24, 26–28, 31–34, 36, 37, 40, 45, 49, 50, 52–55, 57–59, 61, 64, 68, 69, 87, 89, 90, 101–103

regular 8
series-parallel 58, 67, 69
simple 10, 11
unicyclic 11, 58, 67, 69
weighted 11, 76

H

Hamiltonian cycle completion . 9, 58, 67, 68
hierarchy 15
hierarchy constraints 8

I

incidence matrix 13
oriented 13, 20, 22

L

Laplacian matrix 7, 14
level planar . 8–10, 16, 17, 26–28, 31–34, 37, 49, 50, 52, 53, 68, 70, 74–81, 84, 85, 87–92, 97, 103
biplanar 16
cyclic 26, 49
level planar subgraph 57
maximum 2, 4, 5, 8, 9, 57–60, 67–72, 75, 102, 103
level planarity 101–103
cyclic 49
level-by-level sweep 55, 60
leveling 4, 14–18, 20, 21, 26, 28, 31–33, 50, 53, 57, 61, 63, 64, 66, 68, 69, 74, 75, 77, 79, 81, 87, 92–95, 97, 101
cyclic 49
proper 49
macro-levels 81
micro-levels 81
non-proper 27
proper 14, 75, 93
line graph 20
linear arrangement 61
cost 61
minimum 7, 61, 97, 98
linear extension 14, 65

M

MAX-BALANCED-SUBGRAPH 53, 55, 56, 101, 102
MAX-CUT 56
MIN-DUMMY-VERTICES 81
MIN-MAKESPAN 81, 83, 84
multi-level crossings
minimum 8, 9, 26, 28, 49, 52, 54, 57–62, 64, 67, 75, 97, 98, 101, 103

O

order
partial 9, 14, 64, 65, 67
topological 14, 62, 63
total 14, 15, 61, 64, 65, 69, 71, 79

P

parameterized line graph . . 20–22, 24, 25

path 11–13, 15, 21, 22, 29, 31, 65, 68, 78, 84, 85, 89, 91
  directed 12, 13, 69–72
  disjoint 11, 68
  Hamiltonian 11
path cover 68
  minimum 68
pattern 87
  level non-planar 87
    minimum 87, 88, 101–103
penalty digraph
  crossing 9, 62
  planarizing 9, 67, 69, 71, 102
PQ-tree 26, 27, 77

S

sifting 60
sign mapping . . . . *see* signature
signature 17, 18, 53
signed graph 10, 17, 18, 53, 102
smothering 84, 85, 88
spanning tree . . . 9, 12, 74–77, 98, 101
  significant . 9, 74–76, 97, 98, 101, 102
sparse matrix form 88, 89
SPQR-tree 95
star 12, 63
subgraph . . 11, 12, 18, 21, 22, 25, 52, 67, 103
  induced 11
  spanning 11, 12
subtree 9, 12, 68, 78, 87, 88
Sugiyama framework 4, 8, 52, 75
suited pairs 54
  natural order 54
  non-trivial 54
  trivial 54
switch . . 17, 18, 31, 33, 34, 36, 37, 45
symmetric vertices 8

T

task scheduling 80
tournament 13, 71
  acyclic 13, 69–72
transitive closure 13, 54
transitive reduction 13, 98
tree . 9, 11, 12, 17, 20, 25, 57, 61, 68, 69, 78, 83, 90, 91, 95, 98
  k-ary 12
  binary 12, 58
  child 12, 50–52
  complete 12, 58
  descendant 12, 78
  directed 12, 74
  free 12, 74
  inner vertex 12, 50, 52
  leaf . 12, 50, 51, 65, 67, 69, 70, 72
  leveled . 21, 22, 25, 57–59, 61, 62, 64, 66–69, 71, 87, 101, 102
  ordered 12, 50
  parent 12

root . 9, 12, 50, 57, 68, 74, 77, 78, 82, 83, 87, 88, 90, 91
tree DAG 9, 74, 77, 79, 81, 87–90

U

unsuited pairs 54

V

vertex 1, 10
adjacent 10
balance 7
degree . . . 8, 10, 12, 54, 58–60, 65, 67, 69, 83
in-degree 9, 12, 77, 79, 81, 84, 85, 87, 89, 90
incident 10
incoming neighbors 12, 77
neighbors . . 11, 39, 40, 43, 44, 52, 63, 64, 89
out-degree 12
outgoing neighbors 12, 71
sink 12, 27
source 9, 12, 15, 27, 71, 74
vertex-exchange graph 18, 20, 21, 24, 25, 37, 77, 101, 102
labeled . 10, 18, 26, 28, 32–34, 36, 37, 39, 40, 45, 49, 53

W

walk 11, 29–31
length 11
windows 60