A Combinatorial Approach to Orthogonal Placement Problems

Gunnar Werner Klau

Universität des Saarlandes
Saarbrücken, Germany

A Combinatorial Approach to Orthogonal Placement Problems

A Combinatorial Approach to Orthogonal Placement Problems

Gunnar Werner Klau

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.–Ing.)
der Naturwissenschaftlich–Technischen Fakultät I
der Universität des Saarlandes

Universität des Saarlandes
Saarbrücken, Germany

Datum des Kolloquiums: ***3. September 2001***

Dekan der Naturwissenschaftlich–Technischen Fakultät I:
Prof. Dr. Rainer Schulze–Pillot–Ziemen

Gutachter:
Prof. Dr. Petra Mutzel*, Technische Universität Wien, Österreich*
Prof. Dr. Kurt Mehlhorn*, Max–Planck–Institut für Informatik, Saarbrücken*

Short Abstract

We study two families of *NP*-hard orthogonal placement problems that arise in the area of information visualization both from a theoretical and a practical point of view. This thesis contains a common combinatorial framework for compaction problems in orthogonal graph drawing and for point-feature labeling problems in computational cartography. Compaction problems are concerned with performing the conversion from a dimensionless description of the orthogonal shape of a graph to an area-efficient drawing in the orthogonal grid with short edges. The second family of problems deals with the task of attaching rectangular labels to point-features such as cities or mountain peaks on a map so that the placement results in a legible map. We present new combinatorial formulations for these problems employing a path- and cycle-based graph-theoretic property in an associated problem-specific pair of constraint graphs. The reformulation allows us to develop exact algorithms for the original problems. Extensive computational studies on real-world benchmarks show that our linear programming–based algorithms are able to solve large instances of the placement problems to provable optimality within short computation time. Furthermore, we show how to combine the formulations for compaction and labeling problems and present an exact algorithmic approach for a graph labeling problem. Often, our new algorithms are the first exact algorithms for the respective problem variant.

Kurzzusammenfassung

Wir betrachten zwei Familien von *NP*–schwierigen orthogonalen Platzierungsproblemen aus dem Bereich der Informationsvisualisierung von einem theoretischen und praktischen Standpunkt aus. Diese Arbeit enthält ein gemeinsames kombinatorisches Gerüst für Kompaktierungsprobleme aus dem Bereich des orthogonalen Graphenzeichnens und Beschriftungsprobleme von Punktmengen aus dem Gebiet der Computer–Kartografie. Bei den Kompaktierungsproblemen geht es darum, eine gegebene dimensionslose Beschreibung der orthogonalen Form eines Graphen in eine orthogonale Gitterzeichnung mit kurzen Kanten und geringem Flächenverbrauch zu transformieren. Die Beschriftungsprobleme haben zur Aufgabe, eine gegebene Menge von rechteckigen *Labels* so zu platzieren, dass eine lesbare Karte entsteht. In einer klassischen Anwendung repräsentieren die Punkte beispielsweise Städte einer Landkarte, und die Labels enthalten die Namen der Städte. Wir präsentieren neue kombinatorische Formulierungen für diese Probleme und verwenden dabei eine pfad- und kreisbasierte graphentheoretische Eigenschaft in einem zugehörigen problemspezifischen Paar von Constraint–Graphen. Die Umformulierung ermöglicht es uns, exakte Algorithmen für die Originalprobleme zu entwickeln. Umfassende experimentelle Studien mit Benchmark–Instanzen aus der Praxis zeigen, dass unsere Algorithmen, die auf linearer Programmierung beruhen, in der Lage sind, große Instanzen der Platzierungsprobleme beweisbar optimal und in kurzer Rechenzeit zu lösen. Ferner kombinieren wir die Formulierungen für Kompaktierungs– und Beschriftungsprobleme und präsentieren einen exakten algorithmischen Ansatz für ein Graphbeschriftungsproblem. Oftmals sind unsere neuen Algorithmen die ersten exakten Algorithmen für die jeweilige Problemvariante.

Acknowledgments

Many people have taught, encouraged, supported, helped, and advised me during the time in which I worked on this thesis. I wish to express my deepest gratitude to all of them.

First of all, I would like to thank my advisor, Prof. Petra Mutzel, for providing a perfect balance of scientific guidance and scientific freedom. Petra has been a great source of motivation and I am grateful to her for having me introduced to the fascinating research areas of graph drawing and map labeling, for teaching me many things about combinatorial optimization, and for guiding my research work that led to this thesis. Setting up our private *Doktorandenseminar* at MPI Saarbrücken and its continuation in Vienna proved to be a very good idea, and I am indebted to all of its participants, in particular to René Weiskircher and Thomas Ziegler, for interesting discussions.

I also wish to thank Prof. Kurt Mehlhorn. Kurt is responsible for the highly enjoyable scientific and international atmosphere at the Max Planck Institute für Informatik in Saarbrücken and an enthusiastic teacher. Much of what I know about algorithms and data structures I learned in Saarbrücken and, in particular, at MPI. I consider it an honor and privilege to have had the possibility to meet so many inspiring people in the algorithm and complexity group of this institute and to do my first research steps in such conducive conditions. In Saarbrücken I experienced how much fun teaching can be, and I owe much of this discovery to the diploma students I had the pleasure to advise. In particular, I am grateful to Karsten Klein, who is also a co-author on a paper in experimental graph drawing and who contributed to the computational study on compaction algorithms.

Particular thanks to Prof. Matteo Fischetti for helpful and enlightening discussions concerning the relation between the zero-one and the extended polytopes, to Prof. Gerhard Woeginger for lessons in complexity theory, to Prof. Andrew V. Goldberg for his negative cycle detection code, and to Alexander Wolff for the real-world labeling data and the support in the conversion to our data format.

Moreover, I have appreciated the possibility of temporarily using the facilities of the Discrete Optimization group at the University of Heidelberg, for which I would like to thank Prof. Gerhard Reinelt and his group.

Further, I like to thank the German Federal Ministry of Research (BMBF) for financially supporting the project “Automatisiertes Zeichnen von Zustandsgraphen”, and Prof. Ulrich Lauther, Siemens AG, for cooperation.

Thanks to my proof-readers René Weiskircher, Thomas Ziegler, Sebastian Leipert, Karsten Klein, Marco Lübbecke, Birgit and Knut Reinert, Hedwig, Ragnar, and Arne Klau, to Solofo Ramangalahy for sharing his TEX wisdom, and to Martin Gruber for technical support.

Very special thanks to my family and to Stéphanie Dagron.

CONTENTS

Chapter I
Introduction

Art, like morality, consists in drawing the line somewhere.
G. K. Chesterton

This thesis analyzes two families of orthogonal placement problems that arise in the area of information visualization. The first family, compaction of orthogonal grid drawings, is concerned with performing the conversion from a dimensionless description of the orthogonal shape of a graph to an area-efficient drawing in the orthogonal grid with short edges. This two-dimensional compaction problem emerges in the last phase of a powerful approach to high-quality orthogonal graph drawing, the topology-shape-metrics scheme. The second family of problems plays an important role in the area of computational cartography and deals with the task of attaching rectangular labels to point-features such as cities or mountain peaks on a map.

It is common to both drawings of graphs and cartographic maps that they convey complex information about relations of objects as a geometric representation. Moreover, the utility of this representation depends on the quality of the layout process. The overall aim is to generate a drawing or a map of maximum readability that is intuitive to understand and use and which effectively communicates the underlying information. As an example, Figure 1.1 shows "good" and "poor" solutions of the orthogonal placement problems dealt with in this thesis.

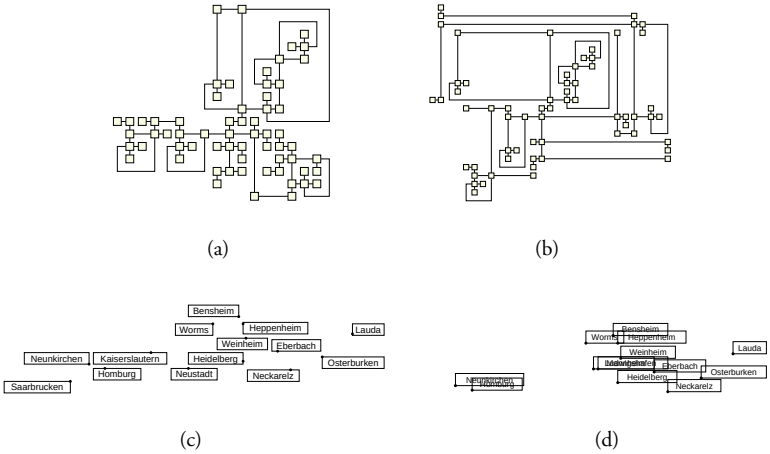


Figure 1.1: Good and poor solutions of orthogonal placement problems

Why are the orthogonal placements in Figure 1.1(a) and (c) better than those of Figure 1.1(b) and (d)? The edges of the right orthogonal drawing are unnecessarily long, confuse the reader, and increase the amount of drawing space needed. The compact drawing on the left has been enlarged by 25% and is still more area-efficient. Due to the better resolution and shorter edges it is superior to the right drawing. Even more obvious reasons make the left labeling a better one than the right one: In Figure 1.1(d), information is lost since not all labels are placed. Furthermore, many of the labels overlap which makes it difficult to extract the necessary information.

A further common characteristic to both areas is that most of the respective problems are computationally hard to solve. Everybody who has tried to draw a graph with 20 vertices by hand knows about the difficulties of finding an aesthetically pleasing drawing. Even if this task can be accomplished, it remains a time-consuming process. The same applies to the placement of labels on a map. Manual map lettering is a tedious and complicated task that takes a lot of time.

In this thesis, we develop a common combinatorial framework for two-dimensional compaction problems in graph drawing and point-feature map labeling problems. The combinatorial formulations allow us to devise exact algorithms that solve large problem instances in short computation time by exploiting the fact that these problems decompose naturally into two largely separate horizontal and vertical problem components. Furthermore, solutions of orthogonal placement problems can be determined by assigning x- and y-coordinates to problem-specific objects (*e.g.*, vertices and bends in orthogonal compaction problems or boundaries of rectangular labels).

Our new approach to these types of orthogonal placement problems is based on a pair of *constraint graphs*—one for each problem component. A constraint graph is a directed graph that represents precedence relations between objects; its arc weights additionally specify these relations. In orthogonal placement problems, the nodes of the horizontal and vertical constraint graphs correspond to the x- and y-coordinates of the problem-specific objects. Weighted directed edges in the graphs represent distance relations between these objects. The idea of the constraint graph-based approach is to set the coordinates of the underlying objects by assigning values to the nodes of the constraint graphs.

A related characterization is used by Bartusch, Möhring, and Radermacher (1988) in the area of scheduling. In scheduling, the nodes of a constraint graph correspond to the jobs and the weighted edges characterize the temporal constraints between these jobs. The authors consider optimization in the set of feasible schedules with resource constraints and time windows. They characterize this set as extensions of a given partial order that satisfy certain order-theoretic properties. Due to the one-dimensional nature of scheduling problems they do not have to consider the relations and dependencies between different constraint graphs. This is exaclty where we focus on in this thesis. The interaction of a pair of constraint graphs will model the interplay of the dimensions in the two-dimensional problems under consideration.

For both the compaction problems and the map labeling problems, we introduce special constraint graphs: a pair of *shape graphs* and a pair of *labeling graphs*. We study these pairs of directed graphs and identify a path- and cycle-based property, *completeness*, that allows us to characterize the circumstances under which a separate computation

of coordinates in each of the constraint graphs leads to feasible solutions. In this case, which may arise for the compaction problems, we will present algorithms that solve the original problems to provable optimality in polynomial time. Moreover, the property of completeness enables us to formulate combinatorial problems that are equivalent to the original orthogonal placement problems. In essence, the new combinatorial formulations ask for an arc set which can be added to the shape or labeling graphs so that the resulting pair of constraint graphs is complete. For some instances of the compaction problems this arc set is unique, and we derive an exact polynomial-time algorithm for the original problem in this case. In most cases, however, several possibilities exist to complete the constraint graphs; it is the combination of these choices that makes the orthogonal placement problems difficult at the combinatorial level.

Yet, our combinatorial reformulations are well-suited to apply integer linear programming techniques. We define problem-dependent polytopes, which correspond to the set of complete extensions, and characterize integral points within these polytopes by integer linear programming formulations. We provide branch-and-bound and branch-and-cut algorithms and present extensive computational studies that show that our new algorithms are able to solve large instances of the *NP*-hard two-dimensional compaction and labeling problems to optimality in short computation time.

One advantage of our common combinatorial framework for compaction and labeling problems is that we can combine them without too much effort in order to solve *graph labeling problems*. Problems in this class combine elements of graph drawing and map labeling problems and, until now, only little research work has been done in that area. We introduce a new problem from this class which arises in the area of automation engineering: drawing labeled state diagrams. We show how to combine our results for the subproblems and provide an exact algorithm for this problem. To our knowledge, this is the first exact algorithm especially designed for solving a graph labeling problem.

1.1 Graph Drawing

In his book on graph theory, Diestel (1997, p.2) writes:

> *The usual way to picture a graph is by drawing a dot for each vertex and joining two of these dots by a line if the corresponding two vertices form an edge. Just how these dots and lines are drawn is considered irrelevant: all that matters is the information which pairs of vertices form an edge and which do not.*

While this statement might apply to a certain extent to the world of a pure graph theorist, it is altogether wrong in case graphs are used to convey information. Many applications in almost all scientific disciplines make use of graphs with the exclusive aim of visualizing the inherent structures of relational information. The following is a selection of applications where graph visualization is especially important:

– *Software engineering.* Today, the developers of large software projects use diagrams during the design, implementation, and documentation phases. The visualizations used include UML-diagrams, class hierarchies, flow diagrams, and subroutine call graphs.

- *Database design.* Entity-relationship models are used to determine the structure and implementation of large databases. The diagrams help to visualize the complex interaction of data and influence the design decisions.

- *Chip layout.* In the area of VLSI design, many problems that occur during the fabrication of logical circuits find their equivalents in graph drawing problems. In particular the two-dimensional compaction problem we are investigating in this thesis plays a role in both areas.

- *Automation engineering.* State diagrams are used for the design and running of control systems such as production controls or robot controls. In addition to the drawing task, rectangular labels, which contain several lines of program code, have to be placed close to the symbols that represent the states of the system. We consider this problem in a separate chapter, where we combine our results for compaction and labeling problems.

- *Economic sciences.* Business processes as well as macro-economic processes are often modeled by graphs. Further, program evaluation and review technique (PERT) charts are important tools in project management, and most organization structures are displayed as hierarchical diagrams.

- *Biochemistry.* Visualizations of biochemical pathways display underlying chronological and causal dependencies as well as hierarchic structures and symmetries.

The area of *automatic graph drawing* is devoted to the development of algorithms that produce geometrical representations of graphs and to the problems that arise within this context. An overview about the research in this area is provided by the books (Di Battista, Eades, Tamassia, and Tollis, 1999b) and (Kaufmann and Wagner, 2001) as well as by the overview article (Eades and Mutzel, 1999) and by the annotated bibliography (Di Battista, Eades, Tamassia, and Tollis, 1994). In this thesis, we focus on *orthogonal drawings*, that is, drawings of graphs in which the edges are represented by sequences of alternating vertical and horizontal line segments. For numerous applications, orthogonality is a convention (*e.g.*, UML-diagrams, circuit layouts, entity-relationship diagrams), and for many other applications orthogonal drawing algorithms produce the best layouts.

Several criteria have been identified to measure the quality of orthogonal drawings:

- The number of *edge crossings* should be as low as possible. This criterion is often considered as the most important, since crossing lines in a diagram confuse the reader. The number of crossings also plays a crucial role in VLSI design: Every wire crossing must be replaced by a so-called *via* or *contact* to change the layer of wires. This has a negative effect on the quality of fabricated chips. It degrades the electrical stability properties, increases the delay, and decreases the percentage of correctly-working chips.

- The same applies to the number of *bends*. Bends occur where horizontal and vertical line segments of edges alternate and decrease the readability of diagrams, since a

bended line is harder to follow. In VLSI, bends are called *jogs* and negatively affect electrical characteristics and the signal delay of a chip.

– Edges should be drawn short to avoid that related objects are placed far away from each other. The *total edge length* as well as the *maximum edge length* of a drawing should be low.

– Saving drawing space is important in many applications and affects the resolution of the drawing when scaling it to fit on a sheet of paper or on a computer screen. The *area* of a drawing is usually defined as the area of the smallest iso-oriented rectangle that encloses the drawing.

– Other criteria include the display of existing *symmetries* of a graph and of *hierarchies* in directed graphs.

Even the restriction to the optimization of only one of the above criteria results in most cases in an *NP*-hard combinatorial optimization problem. Furthermore, optimality with respect to one criterion may exclude optimality with respect to another, see Figure 1.2.

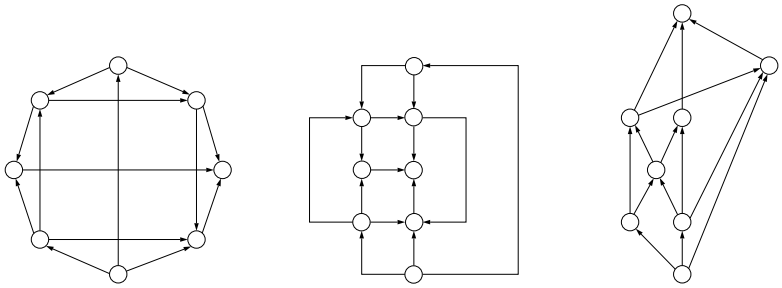


Figure 1.2: Drawing of a graph according to different optimization goals: (a) maximal symmetry, (b) crossing-optimal on the orthogonal grid, (c) hierarchical

Due to the conflicting criteria, tradeoffs cannot be avoided. For orthogonal drawings, the following ranking of aesthetic criteria is widely accepted: The primary goal is to minimize the number of crossings; ideally a graph should be drawn without any crossing edges at all. Secondly, the number of bends should be as low as possible. Finally, besides few crossings and bends, the edges should be short in the drawing, which also leads to good area bounds.

1.1.1 The Topology-Shape-Metrics Scheme

According to the above ranking of criteria, the *topology-shape-metrics scheme*, introduced in (Batini, Nardelli, and Tamassia, 1986) and (Tamassia, Di Battista, and Batini, 1988), leads to the best results in applications of orthogonal graph drawing. The algorithmic

scheme divides the drawing task into three phases and exploits the fact that the class of graphs which admit crossing-free drawings in the plane is well-studied. Graphs that belong to this class are called *planar graphs* and have several useful properties of which specialized drawing algorithms take advantage.

The first phase of the scheme aims at minimizing the number of crossings and is also referred to as the *planarization method*. The method identifies a small set of edges whose removal results in a planar subgraph. Finding a smallest of such sets is an *NP*-hard problem as shown by Liu and Geldmacher (1977). However, for instances of graphs with up to 100 vertices the *maximum planar subgraph* can be found with a branch-and-cut algorithm that exploits the structure of an associated zero-one polytope (Mutzel, 1994; Jünger and Mutzel, 1996). Besides, heuristics can identify large planar subgraphs.

In a second step, the planarization method computes a *planar embedding* for the planar subgraph that fixes the *topology* of this graph. Note that a crossing-free drawing of a graph divides the plane in regions, also referred to as the *faces* of the planar graph. In essence, the topology determines the order of edges around these faces. Generally, the set of different embeddings of a planar graph can be exponential in size, but linear time algorithms exist to construct one planar embedding (Chiba, Nishizeki, Abe, and Ozawa, 1985; Mehlhorn and Mutzel, 1996). The next step of the planarization method consists of reinserting the temporarily deleted edges at the combinatorial level into the existing planar embedding so that the number of crossings is low. In order to maintain a planar graph, every crossing is replaced by an artificial vertex. For one edge, the problem of creating a minimum number of artificial vertices can be solved in linear time, even if the embedding of the subgraph is allowed to change (Gutwenger, Mutzel, and Weiskircher, 2001). In the more general case, in which more edges have to be reinserted, shortest-path based heuristics manage to introduce a low number of crossings. Ziegler (2001) reports on recent progress concerning the crossing minimization problem and the planarization method.

At this stage of the topology-shape-metrics scheme, a non-planar input graph is represented by a planar auxiliary graph. The second phase of the scheme, *orthogonalization*, deals with determining the *orthogonal shape* of the resulting drawing. The optimization goal of this phase is to minimize the number of bends that occur along the edges of the drawing. While it is *NP*-hard to minimize this number over all embeddings of a planar graph (Garg and Tamassia, 1994), the problem can be elegantly solved for a fixed embedding by reducing it to a minimum-cost flow problem as shown by Tamassia (1987). Bertolazzi, Di Battista, and Didimo (2000) present a branch-and-bound algorithm to solve the problem over all embeddings and obtain optimal solutions in reasonable time for small to medium-sized graphs. Weiskircher (2002) investigates the same problem with methods from polyhedral combinatorics and reports similar results.

1.1.2 The Compaction Phase

The output of the orthogonalization phase is a so-called *orthogonal representation* that contains the necessary information about the topology and the orthogonal shape of the drawing. Nevertheless, the description is dimensionless and coordinates still have to be

assigned to the vertices and bends of the drawing. The *compaction phase* of the topology-shape-metrics scheme deals with the transformation of an orthogonal representation into an orthogonal drawing with small total edge length or little area. Again, this is an *NP*-hard problem as shown by Patrignani (1999). Previous algorithmic research for this problem can be divided into constructive and improvement heuristics: In the context of VLSI design, Vijayan and Wigderson (1985) present a first constructive heuristics that can be used for the compaction problem in graph drawing and has quadratic running time. Independently, Tamassia (1987) and Hoffmann and Kriegel (1988) improve this result and present linear-time methods which are based on rectangular dissection of the original orthogonal representation. Bridgeman, Di Battista, Didimo, Liotta, Tamassia, and Vismara (2000) extend these techniques by introducing the concept of turn-regularity. However, the results produced by the constructive heuristics still admit room for considerable improvement. The compression-ridge method (Akers, Geyer, and Roberts, 1970; Dai and Kuh, 1987) and graph-based compaction techniques (*e.g.*, Hsueh, 1979) originate in VLSI and constitute improvement heuristics for the compaction problem. They consider the one-dimensional subproblems of reducing the horizontal or vertical edge lengths. In many cases, iterative usage of these heuristics with alternating direction in a one-dimensional compaction scheme yields considerable improvement. Schlag, Liao, and Wong (1983) and Kedem and Watanabe (1984) present the only exact algorithms for two-dimensional area minimization in the area of VLSI design.

1.1.3 A New Approach to Two-Dimensional Compaction

The key idea of our new approach to the two-dimensional compaction problem as it appears within the topology-shape-metrics scheme is to translate it into an equivalent combinatorial problem involving a pair of constraint graphs. Investigating combinatorial properties of these graphs leads to new algorithms that can solve large instances of the compaction problems to optimality in short computation time. Based on the observation that we can treat the horizontal and vertical direction to a great extent separately, each directed graph corresponds to one such direction. We introduce a generic concept, the *placement graphs*, a pair of constraint graphs that is able to determine the layout of a drawing for a given instance of the compaction problem through an assignment of values to its nodes. The placement graphs generalize, among others, the *layout graphs*, that have already been used in graph-based one-dimensional compaction heuristics.

We investigate how the placement graphs must interact in order to develop a combinatorial characterization of the compaction problem. Thereby, we exploit the fact that, due to the given shape, many relative positions of vertices, edges, and bends are already determined. We introduce a specialization of placement graphs, the so-called *shape graphs*, that reflect the orthogonal shape of the input. Moreover, we identify a central property of placement graphs, *completeness*, that is based on paths and cycles in the two directed graphs and forms the link between the otherwise unconnected horizontal and vertical constraint graph. Substantially, a pair of complete placement graphs consists of two acyclic constraint graphs in which each pair of objects is separated by one of four paths. Each of these paths corresponds to one of the four possible relative placements of a pair of objects

in two dimensions. We show that, for complete placement graphs, the compaction problem reduces to two separate one-dimensional problems for which optimal solutions lead to an optimal solution of the problem in two dimensions.

The shape graphs are uniquely determined by the given orthogonal representation. In case of complete shape graphs, we can solve the two-dimensional compaction problem to optimality in polynomial time. We also consider known compaction heuristics in view of the new combinatorial formulation and relate the respective placement graphs to the concept of completeness. We show that shape graphs corresponding to auxiliary orthogonal representations, as produced by rectangular dissection methods, are complete. Furthermore, we introduce the *visibility graphs* that are used within graph-based improvement heuristics, and demonstrate that they are close to completeness: A feasible assignment of values to the nodes in one of the visibility graphs results in a feasible solution of the compaction problem.

We also investigate a popular one-dimensional compaction scheme and demonstrate that instances exist for which a linear number of alternating one-dimensional compaction steps is necessary. Moreover, we show that algorithms within the scheme do not approximate the compaction problem by a constant factor.

Every orthogonal drawing uniquely determines a pair of visibility graphs. The observation that every such pair contains the pair of shape graphs of the appropriate orthogonal representation leads to the combinatorial reformulation of the two-dimensional compaction problem. In a way, we are looking for the perfect pair of visibility graphs that results from adding arcs to the shape graphs. We identify a set of *potential additional arcs* and show that the set of *complete extensions* that results from adding certain subsets of potential arcs to the shape graphs is in one-to-one correspondence to the feasible solutions of the original problem. It is the choice of potential arcs that makes the compaction problem difficult at the combinatorial level.

However, we can characterize those shape graphs which admit a unique extension and solve the compaction problem in polynomial time for these instances. For general instances, our combinatorial reformulation allows a translation into an integer linear program. We characterize the integral points within the *compaction polytope* by different classes of inequalities and integrality constraints. Then we show that feasible solutions of the resulting integer linear program correspond to feasible solutions of the combinatorial equivalent and vice versa, thus providing orthogonal drawings for the given orthogonal representation. This enables us to optimize over the set of feasible orthogonal drawings for a given instance, and we present both a branch-and-bound and a branch-and-cut algorithm to solve the two-dimensional compaction problem to optimality.

We test our implementations on a large set of widely used benchmark-graphs from different test-suites, including a set of 11,582 graphs arising from real-world applications. Our extensive computational study shows that we can solve all real-world problem instances in short computation time.

1.2 Map Labeling

Map labeling problems attract many researchers in computer science. On the one hand, this is due to its numerous applications, *e.g.*, in cartography, geographic information systems, point pattern analysis, spatial statistics, and graphical interfaces. On the other hand, many combinatorial optimization problems with beautiful mathematical properties appear in this area.

The growing amount of data for which informational graphics have to be produced leads to an increasing need for automatic labeling procedures. Due to the complexity of the underlying problems, manual map labeling is a tedious and time-consuming task. In addition to the classical problem of labeling a two-dimensional cartographic map, labeling problems arise in geographic information systems, navigation systems and fully automatically generated technical maps where manual labeling is impossible.

A major problem in map labeling is the *point-feature label placement* in which the task is to place labels adjacent to point features so that only a few or no labels overlap. These features may be cities, mountain peaks or points in a plot which represent statistical data. Besides point-features, also area features, like countries and seas, or line features, like streets and rivers, have to be labeled. In this thesis, we focus on point-feature labeling and restrict the labels to be iso-oriented axis-parallel rectangles. We provide a detailed overview of previous work on rectangular point-feature labeling; the bibliography (Wolff and Strijk, 2001) is a good starting point for literature on other models.

Several criteria have been developed that distinguish a high-quality labeling from a poor one:

- On a good map the placement of labels is as unambiguous as possible. It is intuitively apparent to the reader of the map which label belongs to which point-feature. This implies that labels are close to the point-features they belong to.

- The information of the labels is legible. Unambiguity alone does not help if the user cannot read the text in the labels.

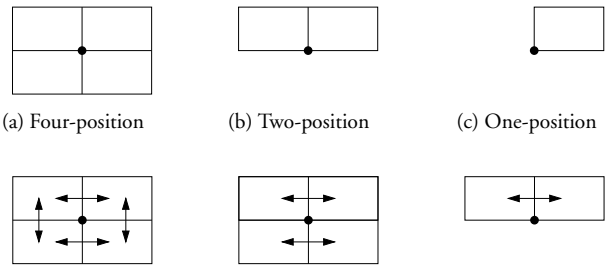
- No or only a few labels overlap. Obviously, overlaps decrease the legibility of a map.

The cartographic literature contains more rules, see, *e.g.*, (Imhof, 1962) and (Yoeli, 1972). Yet, the overall aim in automatic map labeling is to devise algorithms that produce labelings of maximum legibility. To be consistent with our notation in graph drawing, we also refer to the above principles as *aesthetic criteria*.

We concentrate on the six different labeling models in Figure 1.3. The *discrete models* or *fixed-position models* displayed in Figure 1.3(a)-(c) allow only a finite number of positions per label. Among the most popular discrete models in practice is the four-position model; the two- and one-position models exist rather for theoretical purposes.[1] More natural than the discrete models are *slider models* which allow a continuous movement of a label around its point-feature, see Figure 1.3(d)-(f).

[1] This thesis concentrates on the models displayed in Figure 1.3 since they are common in the computer science literature. In practice, cartographers often use eight-position or other discrete models.

Clearly, different bounds hold for the maximum number of non-overlapping labels that can be placed close to each point-feature in the different models: At maximum four labels per point-feature are possible in the four-slider, two-slider and four-position model, maximally two in the one-slider and two-position, and at most one in the one-position model.



(a) Four-position (b) Two-position (c) One-position

(d) Four-slider (e) Two-slider (f) One-slider

Figure 1.3: Axis-parallel rectangular labeling models. A label can be placed in any of the positions indicated by the rectangles and can slide in the directions of the arcs

An instance of a labeling problem consists of a set of point-features, information about the label sizes, and a mapping from labels to point-features. In general it is not possible to place all the given labels in their original size without any overlap. The literature suggests several possibilities to deal with this problem; among these are decreasing the size of the labels to allow a placement of all labels without any overlap, and keeping the sizes of the labels fix while looking for the maximum number of labels that can be placed. The first possibility is referred to as the *label size maximization problem*, the second one as the *label number maximization problem*. A third variant is to tolerate overlaps but to aim at a low number of mutually overlapping labels: the *label overlap minimization problem* asks for a labeling in which the number of overlaps is minimum. In this thesis, the main focus is on the label number maximization problem; the underlying concepts, however, result also in powerful approaches to the other two problems.

1.2.1 A New Approach to Map Labeling Problems

Rectangular map labeling problems share the same characteristics of orthogonal placement problems as the two-dimensional compaction problems. This motivates us to use a similar approach as for the problems in graph drawing. We reuse our idea of developing combinatorial translations of the original problems, and demonstrate that this leads to new powerful algorithms for a variety of labeling problems.

Again, we associate a pair of constraint graphs with problem instances; in the case of point-feature labeling problems these are the *labeling graphs*. The key idea is the same as for the two-dimensional compaction problems: If these graphs satisfy certain path- and cycle-based properties, we can produce a solution for the original problem by separately

assigning values to the nodes of the constraint graphs. These values correspond to x- and y-coordinates of the solution and determine the placement of labels.

For a given instance of a labeling problem we construct a special pair of labeling graphs. We introduce different kinds of arcs whose presence satisfy necessary properties of feasible placements of labels: The *fixed distance arcs* ensure that the relative position between the point-features remains fix. *Label size arcs* perform a similar task and guarantee that every label is represented by a rectangle of width and height as described in the input. By introducing the *proximity arcs* we determine the rectangular region around the appropriate point-feature in which a label can be placed. In order to exclude that a label covers the point-feature it belongs to, we define the *boundary arcs* which are inverse to the proximity arcs. Unlike the previously introduced types of arcs, the boundary arcs belong to the class of *potential arcs* and influence the labeling model. Each discrete or slider model corresponds to requirements on subsets of boundary arcs which have to be present in the labeling graphs. We define a second type of potential arcs in order to control the overlaps between labels. The *label separation arcs* make sure that pairs of labels do not overlap in a placement.

The potential additional arcs fulfill the same task as for the compaction problems. We can restate the pure labeling problem in which all labels have to be placed without scaling and overlaps as the identification of a subset of potential arcs that satisfies the following two properties. First, the set of chosen boundary arcs must comply with the appropriate labeling model. The second property extends the notion of *completeness* as defined for the compaction problems: At least one label separation arc has to be chosen for each label pair, and adding the entire set of chosen potential arcs to the labeling graphs must not induce directed cycles of positive weight. We show that the combinatorial reformulation is equivalent to the pure labeling problem by establishing a one-to-one correspondence between feasible solutions. Furthermore, we demonstrate how to adapt the new combinatorial problem to result in equivalent formulations of the label number maximization problem and the label overlap minimization problem. We find it remarkable that our new approach is independent of the labeling model and results in discrete formulations even if the problems are of continuous nature as in the slider models.

The combinatorial formulation for the pure labeling problem admits a straightforward characterization as a zero-one polytope through an incidence vector for the set of potential arcs. We provide an integer linear programming formulation for this polytope by describing feasible solutions of the combinatorial version of the pure labeling problem with classes of inequalities and integrality constraints. One class of inequalities are the so-called *positive cycle inequalities*. We investigate the corresponding separation problem and show that it is *NP*-complete by a reduction from the shortest weight-constrained directed path problem. We present an extended formulation that evades the class of positive cycle inequalities. However, the prize for omitting the cycle inequalities are additional continuous variables that are linked to the binary variables with a "big M" approach.

Our integer linear programs for the label number maximization problem are not as straightforward as for the pure labeling problem. We present a first formulation with an additional binary variable vector that represents the decision to place or not to place a label. We integrate the new variables in the existing inequalities and show that feasible

solutions of the resulting formulations correspond to an overlap-free labeling for a set of labels of appropriate size. This integration works for both the zero-one and the extended formulation. In a second formulation we manage to eliminate the newly introduced decision variables by a substitution step. However, we have to add additional inequalities to adjust the objective function.

The integer linear programming formulations give rise to new algorithms for the labeling problems. We present branch-and-bound and branch-and-cut algorithms and an iterative branch-and-bound scheme for the zero-one and extended formulations. The algorithms work in all labeling models and are the first exact algorithms for the continuous slider models. Preliminary evaluation identifies the iterative branch-and-bound scheme for the zero-one formulation as the most suitable method to solve practical instances of the label number maximization problem. We provide extensive computational experiments in which we test our new algorithm on a large set of benchmark data. The results show that the exact algorithms are competitive and produce optimal solutions for large instances in reasonable computation time.

1.3 Combining Drawing and Labeling: Graph Labeling

Combining graph drawing and map labeling problems results in an interesting new problem class. We define *graph labeling problems* as problems from the area of graph drawing in which subsets of vertices and edges have to be labeled. Unlike in map labeling where the position of the objects is specified in the input, the coordinates of vertices and edges in an instance of a graph drawing problem have yet to be determined and thus create additional degrees of freedom. Little research work exists on this subject.

We consider a special graph labeling problem that occurs in the area of automation engineering: drawing labeled state diagrams. State diagrams represent control systems such as production or robot controls. They are used in the design and documentation phase of these systems and consist of state nodes, transition nodes, transition edges, and rectangular labels. A label contains several lines of program code and belongs to a state or transition node. The task is to provide an area-efficient drawing of the underlying graph and to place all labels according to the criteria in map labeling.

We propose an approach that combines the compaction problem of the topology-shape-metrics scheme for orthogonal graph drawing with a map labeling problem. This enables us to merge the combinatorial frameworks for the subproblems. We introduce the *shape and labeling graphs* that combine the features of the respective pairs of constraint graphs for the compaction and labeling problems. We proceed as for the subproblems and extend the concept of *complete extensions* to the shape and labeling graphs. Again, we manage to develop a combinatorial reformulation for which we present an equivalent integer linear programming formulation. We present a branch-and-bound algorithm to solve this formulation, and, to our knowledge, provide the first exact algorithm especially designed for graph labeling problems.

1.4 Overview

Chapter 2 provides basic definitions and notations from the areas of graph theory, graph drawing, linear programming, and combinatorial optimization. Moreover, we present the generic branch-and-bound and branch-and-cut algorithms to solve combinatorial optimization problems.

We formally introduce constraint graphs and several of their properties in Chapter 3. In particular, we investigate the circumstances under which feasible and optimal values can be assigned to the nodes and present algorithms which accomplish this task. We consider various special cases that give rise to more efficient algorithms. This chapter contains the necessary tools to solve the orthogonal placement problems in case only one constraint graph is involved and thus provides the basis for the algorithms developed in the following chapters.

We study the two-dimensional compaction problem in Chapter 4. Section 4.1 formally introduces the problem and contains complexity results. In Section 4.2, we develop the new combinatorial characterization by introducing the placement graphs, the shape graphs, and the property of completeness. We present a central theorem that relates complete placement graphs to feasible and, in particular, optimal solutions of the compaction problem in orthogonal graph drawing. A large part of the section is dedicated to the review of constructive and improvement heuristics in the light of the newly introduced combinatorial characterization. Moreover, we study a widely used one-dimensional compaction scheme: We demonstrate that instances exist for which the scheme takes a linear number of steps and show that iterative one-dimensional compaction does not lead to approximation algorithms for the two-dimensional compaction problem. We conclude the section by presenting our combinatorial reformulation and by proving its equivalence to the original problem. Section 4.3 describes exact algorithms for the combinatorial version. We introduce the concept of uniquely completable representations and present an exact polynomial-time compaction algorithm for this case. The algorithm is also useful in the general case, since it identifies the potential arcs among which we must choose in order to construct a complete extension of the shape graphs. We provide integer linear programming formulations for this task and present a branch-and-bound algorithm that solves the two-dimensional compaction problem for general orthogonal representations to provable optimality. Moreover, we study the class of cycle inequalities, show how to solve the corresponding separation problem, and extend the algorithm by adding cutting planes. Finally, we discuss related work in the area of VLSI design. We conclude the chapter with Section 4.4 where we present an extensive computational study of orthogonal compaction algorithms.

The subject of Chapter 5 is map labeling. We present a new combinatorial characterization of various labeling problems in six different labeling models. Our combinatorial approach leads to new algorithms that can solve large instances of the problems to optimality within short computation time. Section 5.1 contains mathematically precise definitions of different labeling problems and presents a comprehensive overview of previous work and complexity results. In Section 5.2, we introduce the labeling graphs and demonstrate how to create this pair of constraint graphs depending on the input data. Further, we state the

combinatorial problem equivalents of the labeling problems. We introduce the labeling polytope in Section 5.3 and characterize its integral points with a zero-one integer linear programming formulation. We study the separation problem for the class of positive cycle inequalities and show that it is *NP*-complete to decide whether a solution of the linear programming relaxation contains a violated inequality from this class or not. We present an extended formulation that avoids the cycle inequalities at the prize of new variables. In this section we also develop our integer linear programs for the label number maximization problem and show the equivalence to the original formulations. In Section 5.4, we derive exact algorithms for different labeling problems. Based on the combinatorial equivalents we present both pure branch-and-bound and branch-and-cut algorithms as well as an iterative branch-and-bound scheme. We provide an implementation of the scheme and test it on a large number of instances of benchmark labeling problems. We describe our experimental results for the label number maximization problem in Section 5.5.

In Chapter 6, we apply the results of the two preceding chapters in order to develop an algorithm for a problem from the class of graph labeling problems. We discuss properties of this new problem class in Section 6.1 and focus on a special graph labeling problem, drawing labeled state diagrams, which we introduce in Section 6.2. In Section 6.3 we derive an algorithm for this problem that results from combining the combinatorial frameworks which we have developed in the chapters on compaction and map labeling. Our detailed studies of the subproblems results in an exact branch-and-bound algorithm for the combined compaction and labeling problem, to our knowledge the first exact algorithmic approach to the class of graph labeling problems.

We conclude with Chapter 7 where we discuss our results and mention possible extensions of the new techniques presented in this thesis. We believe that our combinatorial characterizations are expansible and suitable to apply them to related problems like, *e.g.*, packing or location problems.

Chapter 2 Preliminaries

This chapter introduces the mathematical and combinatorial structures, properties and techniques that are used in this thesis and provides notational conventions. The first two sections are dedicated to graphs and their representation in the plane: While Section 2.1 presents fundamental graph-theoretical concepts, Section 2.2 introduces terminology from planarity theory and the area of graph drawing. Many algorithms in the following chapters are of combinatorial nature and involve methods from linear programming and polyhedral combinatorics: Section 2.3 introduces the basics of linear programming theory, and Section 2.4 presents generic methods that use these concepts to solve linear combinatorial optimization problems.

First, we define some notation in linear algebra: We denote the set of real, non-negative real, rational, non-negative rational, and integer numbers by the symbols $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{Q}$, $\mathbb{Q}_+$, and $\mathbb{Z}$, respectively. Let $E = \{e_1, e_2, \ldots, e_n\}$ be a finite *ground set* whose order is reflected by the indices, and let X be an arbitrary set. Then X^E denotes the vector space in which the components of each vector are indexed by the members of E. In case $E = \{1, \ldots, n\}$, we also write X^n. We always consider vectors as column vectors and write row vectors as transposed column vectors x^T. The symbol $\mathbb{1}$ denotes the vector in which all components are equal to one.

The *power set* of a set S is the set of subsets of S, and we write $\mathcal{P}(S)$. The subset of $\mathcal{P}(S)$ whose members have exactly d elements will be denoted as $\binom{S}{d}$.

2.1 Graph Theory

A *graph* is a pair of two disjunct finite sets (V, E). The first set is non-empty, and its elements are the *vertices*, the second set contains the *edges*. An edge $e \in E$ is a pair of two vertices $v \in V$ and $w \in V$. Both v and w are *incident* to e and *adjacent* to each other. In this case, we also refer to v and w as *neighbors*.

We distinguish between *directed* and *undirected* graphs. In *undirected graphs* edges are unordered pairs $E \subseteq \binom{V}{2}$, and we write $e = (v, w) \in E$.

The edges in *directed graphs* or *digraphs* are ordered pairs of vertices, thus $E \subseteq V \times V$. As for undirected graphs we write $e = (v, w)$ for a member of this set. To avoid confusion, we refer to the vertices of a digraph $D = (N, A)$ as *nodes* and to its edges as *arcs*. We allow parallel arcs, *i.e.*, the directed edges of a digraph form a multiset; these graphs are also called *multigraphs*. Graphs without multiple edges are *simple graphs*. We call i the *source* of an arc (i, j) and j its *target*. If the source of an arc a is identical to its target,

we say that a is a *loop*. *Weighted digraphs* (D, A, w) have a weight vector $w \in \mathbb{Q}^A$ or a weight function $w : A \to \mathbb{Q}$ associated with their arcs, we refer to the weight of an arc $a \in A$ by w_a or by $w(a)$.

Let $G = (V, E)$ and $G' = (V', E')$ be two undirected or two directed graphs. We define $G \cup G' = (V \cup V', E \cup E')$ and $G \cap G' = (V \cap V', E \cap E')$. If $V \cap V' = \varnothing$, then G and G' are *disjoint*. If $V \subseteq V'$ and $E \subseteq E'$, then G is a *subgraph* of G', and G' is a *supergraph* of G, we also write $G \subseteq G'$. Let V_E be the vertices incident to the edges in E. If $V = V_E$, we say that G is *induced* by E.

A *directed path* of *length* k is a non-empty digraph $P = (N_P, A_P)$ with $k+1$ distinct nodes $N_P = \{n_0, \ldots, n_k\}$ and k arcs

$$A_P = \{(n_0, n_1), (n_1, n_2), \ldots (n_{k-1}, n_k)\} \ ,$$

the definition for undirected graphs is analogous. The path P *links* its endnodes n_0 and n_k. We also write $n_0 \xrightarrow{*} n_k$ or refer to P by the sequence of its arcs.

The *weight of a path* in a weighted digraph is the sum of its arc weights, *i.e.*,

$$w(P) = \sum_{i=0}^{k-1} w\big((n_i, n_{i+1})\big) \ .$$

A *positive path* is a path $P = n_0 \xrightarrow{*} n_k$ with weight $w(P) > 0$; in this case we also write $n_0 \xrightarrow{+} n_k$.

A *cycle* C is a graph that consists of a directed path $P = n_0 \xrightarrow{*} n_k$ of length $k \geq 1$ and an additional arc (n_k, n_0). Its *weight* is $w(C) = w(P) + w(a)$, and its *length* is $k + 1$. In case $w(C) > 0$, we say that C is a *positive cycle*. A graph is *acyclic* if none of its subgraphs is a cycle.

The *transitive closure* $D^* = (N, A^*)$ of a directed graph $D = (N, A)$ contains an arc for each path in D, *i.e.*,

$$A^* = \{(i, j) \in N \times N \mid i \xrightarrow{*} j \subseteq D\} \ .$$

We also call the arc set A^* the transitive closure of A.

A graph is *connected* if a path between every pair of vertices exists. A connected subgraph is also called a *connected component*; a *maximally connected component* is a connected component of maximum size.

In an undirected graph $G = (V, E)$, the *degree* of a vertex v is the number of its incident edges, *i.e.*,

$$\delta(v) = |\{w \mid (v, w) \in E\}| \ .$$

A *four-graph* is a graph in which the maximum vertex degree does not exceed four. The number of edges in connected four-graphs $G = (V, E)$ is bounded from two sides, it holds

$$|V| - 1 \leq |E| \leq 2|V| \ .$$

For directed graphs $D = (N, A)$, we distinguish between the *indegree*, the *outdegree* and the *net degree* of a node j, more precisely

$$\begin{aligned}\delta_{\text{in}}(j) &= |\{i \mid (i,j) \in A\}| \\ \delta_{\text{out}}(j) &= |\{k \mid (j,k) \in A\}| \\ \delta_{\text{net}}(j) &= \delta_{\text{in}}(j) - \delta_{\text{out}}(j) \ .\end{aligned}$$

2.2 Graph Drawing

A *drawing of a graph* $G = (V, E)$ is a function Γ which maps the vertices V to distinct points of a surface S and the edges E to simple open curves in S that link the images of their incident vertices. Note that according to this general definition edges are allowed to cross and overlap. In the following, we assume that the surface is the two-dimensional Euclidean plane $\mathbb{R}^2$ and that G is connected. A drawing of a directed graph is *upward*, if the images of arcs are monotonically non-decreasing in one common direction.

A graph $G = (V, E)$ is *embeddable in the plane*, if a drawing for G exists in which the images of the edges E are non-intersecting. We call such a drawing an *embedding* and a graph which is embeddable a *planar* graph. In the case that a planar graph is also a four-graph, we call it *four-planar*. If a graph admits an *upward embedding*, then it is *upward planar*. We refer to an embedded graph in the plane as a *plane graph*.

In this thesis, we restrict our focus to embeddings of planar graphs. However, the techniques we are going to present are also suitable for non-planar graphs by employing the *planarization method*. Every graph can be represented by a planar auxiliary graph in which artificial vertices correspond to crossing edges. In this way, an embedding for the auxiliary graph gives rise to a drawing for the original graph.

Note that a plane graph induces a partition of the plane into a set of topologically connected regions, also referred to as the *faces* of the plane graph. Exactly one of the faces is unbounded and is called the *external face* f_0. The numbers of vertices, edges, and faces F of a plane graph are related by

$$|V| - |E| + |F| = 2 \ .$$

With each edge in $e = (v, w) \in E$ we associate two directed *half-edges* $\vec{e_1} = (v, w)$ and $\vec{e_2} = (w, v)$. We refer by $\vec{E}$ to the half-edges corresponding to E. Observe that a plane graph determines a circular ordering of half-edges in each face. This gives rise to a topological description of plane graphs by specifying the order of half-edges for each face. A *planar representation* P of a planar graph G is a set of circularly ordered sets $P(f_i)$ for $i = 0, \ldots, |F| - 1$. Each set $P(f_i)$ contains the cycle of half-edges that define the boundary of face f_i and have the face to their left. The *size* $|f|$ of a face f is the number of half-edges in $P(f)$. If the two half-edges that correspond to the same edge e bound the same face, the edge e is called a *bridge*.

Planar representations are equivalence classes of plane graphs and describe the topology of crossing-free drawings of a planar graph.

Orthogonal grid embeddings play an important role in this thesis. Therefore, we provide a separate definition.

Definition 2.1 (Orthogonal grid embedding). An *orthogonal grid embedding* Γ of a four-planar graph $G = (V, E)$ is an embedding with the following additional properties

1. The embedding Γ maps each vertex $v \in V$ to a grid point, *i.e.*, $\Gamma(v) \in \mathbb{Z}^2$.
2. The embedding Γ maps edges to paths in the grid, *i.e.*,

$$\Gamma(e) \subseteq \{(x, y) \in \mathbb{R}^2 \mid x \in \mathbb{Z} \text{ or } y \in \mathbb{Z}\} \ .$$

Note that, according to Definition 2.1, edges are drawn as sequences of horizontal and vertical line segments; *bends* occur where the path changes its direction in the grid. The number of bends of an edge is the number of these changes, the number of bends of an orthogonal grid embedding is the sum of the bends of the edges.

Definition 2.2 (Simple orthogonal grid embedding). An *orthogonal grid embedding* Γ is *simple* if its number of bends is zero.

Observe that by replacing each bend by an artificial vertex of degree two we can transform every orthogonal grid embedding of a four-planar graph into a simple orthogonal grid embedding of an auxiliary graph. Figure 2.1 shows the relation.

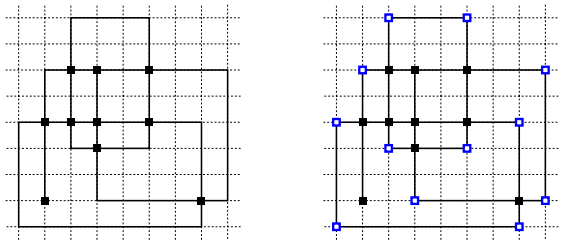


Figure 2.1: An orthogonal grid embedding (a) and its simple counterpart (b)

Unless otherwise stated, we assume that orthogonal grid embeddings are simple. We will often write Γ_x and Γ_y for the horizontal and vertical coordinates of a simple orthogonal grid drawing. Furthermore, if the context is clear, we will also refer to orthogonal grid embeddings as grid drawings or just drawings.

Note that for simple orthogonal grid drawings we can omit the second property of Definition 2.1. Since no bends occur along the edges, the drawing Γ maps an edge $e = (v, w)$ to the straight line segment from $\Gamma(v)$ to $\Gamma(w)$. Hence, it suffices to specify the coordinates of the vertices to characterize a simple drawing.

Let $\Gamma = (\Gamma_x, \Gamma_y)$ be a simple orthogonal grid embedding of a four-planar graph $G = (V, E)$. The embedding induces a direction on the underlying half-edges $\vec{E}$ and

partitions them into the sets $\vec{E}_r$, $\vec{E}_l$, $\vec{E}_u$, and $\vec{E}_d$ that contain the half-edges pointing right, left, up, and down, respectively. It also partitions the edge set into a set of horizontal edges E_h and a set of vertical edges E_v.

The *length* of an edge $e \in E$ is

$$l(e) = \begin{cases} \Gamma_x(w) - \Gamma_x(v) & e = (v,w) \in E_h, (v,w) \in \vec{E}_r \\ \Gamma_y(w) - \Gamma_y(v) & e = (v,w) \in E_v, (v,w) \in \vec{E}_u \end{cases} .$$

The *total edge length* of the embedding is

$$L(\Gamma) = \sum_{e \in E} l(e) \ ,$$

and its *maximum edge length* is

$$l_{\max}(\Gamma) = \max_{e \in E} l(e) \ .$$

The *width* and *height* of the embedding are

$$\begin{aligned} W(\Gamma) &= \max_{v \in V} \Gamma_x(v) - \min_{v \in V} \Gamma_x(v) \\ H(\Gamma) &= \max_{v \in V} \Gamma_y(v) - \min_{v \in V} \Gamma_y(v) \ , \end{aligned}$$

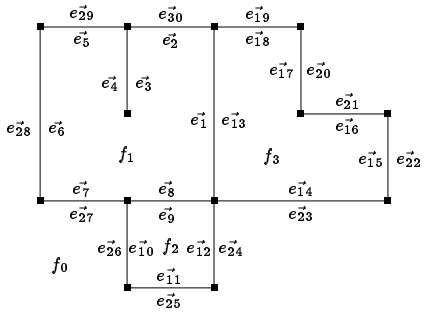
and its area is the area of the smallest iso-oriented covering rectangle

$$A(\Gamma) = W(\Gamma) \cdot H(\Gamma) \ .$$

The concept of *orthogonal shape* of a drawing will play an important part in this thesis. Different notions exist in the literature which is due to the variety of areas in which orthogonal shapes occur, *e.g.*, VLSI design, graph theory, and graph drawing.

We define the term as an equivalence class of simple orthogonal grid embeddings, refining the notion of topology: Two drawings have the same shape if one can be transformed to the other by translation, rotation, and modifying the lengths of the edges. In the area of graph drawing, and in particular within the topology-shape-metrics scheme, the most common way to characterize the shape of an orthogonal embedding is by means of an *orthogonal representation*. Orthogonal representations extend the planar representations by additionally specifying for each half-edge $\vec{h}$ the type and order of bends that occur along $\vec{h}$ and the angle it forms with the following half-edge inside the appropriate face. Again, the implicit assumption of a mapping from bends to artificial vertices allows us to focus on simple orthogonal representations.

Definition 2.3 (Simple orthogonal representation). A *simple orthogonal representation* extends the planar representation $P(f_i), i = 0, \ldots |F| - 1$ of a four-planar graph $G = (V, E)$ by an additional function $a : \vec{E} \to \{90, 180, 270, 360\}$. Let $\vec{h}$ be a half-edge on the boundary of a face f, and let $\vec{i}$ be the following half-edge on the cycle $P(f)$. The value $a(\vec{h})$ defines the angle that occurs between $\vec{h}$ and $\vec{i}$ in f.



i	$a(\vec{e_i})$	i	$a(\vec{e_i})$	i	$a(\vec{e_i})$	i	$a(\vec{e_i})$
1	90	9	90	13	90	19	270
2	90	10	90	14	90	20	90
3	360	11	90	15	90	21	270
4	90	12	90	16	270	22	270
5	90			17	90	23	90
6	90			18	90	24	270
7	180					25	270
8	90					26	90
						27	270
						28	270
						29	180
						30	180

$$
\begin{aligned}
P(f_1) &= (\vec{e_1}, \ldots, \vec{e_8}) \\
P(f_2) &= (\vec{e_9}, \ldots, \vec{e_{12}}) \\
P(f_3) &= (\vec{e_{13}}, \ldots, \vec{e_{18}}) \\
P(f_0) &= (\vec{e_{19}}, \ldots, \vec{e_{30}})
\end{aligned}
$$

Figure 2.2: A simple orthogonal representation

Figure 2.2 illustrates the definition. Note that already the orthogonal shape defines the partitions of the edges E into horizontal and vertical edges and of the half-edges $\vec{E}$ into those pointing right, left, up, and down, respectively.

We treat planar and orthogonal representations as extensions of the underlying four-graphs and will often abuse the notation by speaking of objects like, *e.g.*, faces, drawings, or even dual graphs of orthogonal representations.

In addition to the description in the orthogonal representation, we will also characterize the shape of faces by means of a bit string in order to perform pattern replacing operations on the shape. Let f be a face, and let $P(f) = (\vec{e_1}, \ldots, \vec{e_{|f|}})$ the set of bounding half-edges, with indices reflecting the circular order of the entries. We define

$$
w_i = \begin{cases} 0 & a(\vec{e_i}) = 90 \\ \varepsilon & a(\vec{e_i}) = 180 \\ 1 & a(\vec{e_i}) = 270 \\ 11 & a(\vec{e_i}) = 360 \end{cases} \qquad \text{for all } i = 1, \ldots, |f| \ .
$$

Then the following circular bit string describes the shape of face f:

$$
S(f) = w_1 w_2 \ldots w_{|f|} \ .
$$

An alternative definition of orthogonal shape is by means of a *rectilinear graph* as introduced in (Vijayan and Wigderson, 1985) and used in (Hoffmann and Kriegel, 1988). We adapt the definition to the notion of half-edges:

Definition 2.4 (Rectilinear graph). A *rectilinear graph* is a triple $G = (V, E, \lambda)$, where (V, E) is a four-planar graph, and

$$
\lambda : V \times V \to \{L, R, D, U, \epsilon\}
$$

is a vertex ordering relation with the properties:

(a) $\lambda\big((v,w)\big) = \varepsilon \quad \Leftrightarrow \quad (v,w) \notin \vec{E}$;

(b) $\lambda\big((v,w)\big) = L \quad \Leftrightarrow \quad \lambda\big((v,w)\big) = R$;

(c) $\lambda\big((v,w)\big) = D \quad \Leftrightarrow \quad \lambda\big((v,w)\big) = U$;

(d) $\lambda\big((v,w)\big) = X \quad \Rightarrow \quad \lambda\big((u,w)\big) \neq X$ for all $u \neq v$, $X \in \{L, R, D, U\}$.

Clearly, rectilinear graphs provide a description of orthogonal shape that is equivalent to orthogonal representations, and one notion can be changed to the other in linear time. To construct the rectilinear graph that corresponds to the orthogonal representation H, we just set the labels according to the partition of half-edges. For the other direction, we set the angles by looking at the labels of consecutive half-edges.

2.3 Linear Programming

Mathematical optimization deals with finding optimal solutions subject to a set of constraints and with respect to an evaluation function. Without loss of generality, we will present the definitions for maximization problems that can be easily transformed into minimization problems in order to obtain the corresponding definitions.

Let $A \in \mathbb{R}^{m \times n}$ be a matrix and let $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ be two vectors. A *linear program* consists of a system $Ax \leq b$ of linear inequalities and a linear *objective function* $c^T x$. A vector $\bar{x} \in \mathbb{R}^n$ with $A\bar{x} \leq b$ is a *feasible solution* of the problem. If no such vector exists, we say that the linear program is *infeasible*. The task in the *linear programming problem* is to find an *optimal feasible solution* with respect to the objective function, *i.e.*, a vector x^* with

$$c^T x^* = \max\{c^T x \mid Ax \leq b\} \ .$$

We also write a linear program as

$$\begin{aligned} \max \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \ . \end{aligned} \tag{P}$$

Duality theory relates to each linear program (P) another linear program

$$\begin{aligned} \min \quad & y^T b \\ \text{subject to} \quad & A^T y = c \\ & y \geq 0 \ . \end{aligned} \tag{D}$$

The first one is called the *primal* problem, the second one is *dual* to the first one or simply its *dual*. The relation is symmetric, *i.e.*, the dual of (D) is again (P). The following theorem states a fundamental result in duality theory which has many algorithmic implications.

Theorem 2.1 (Duality theorem of linear programming, *e.g.*, (Schrijver, 1986)). *Let* (P) *and* (D) *be linear programs which are dual to each other.*

(a) If (P) *and* (D) *have feasible solutions, then they have optimal solutions of identical objective function values.*

(b) If (P) *is infeasible, then* (D) *is either infeasible or unbounded.*

(c) If (P) *is unbounded, then* (D) *is infeasible.*

General linear programs can be solved with an efficient method, the *simplex algorithm*, and are solvable in polynomial time. For numerous practical applications elegant linear programming formulations exist of which feasible solutions correspond to feasible solutions of the original problem. Yet, many natural formulations impose integrality constraints on the set of feasible solutions. Such a formulation is called an *integer linear program* or an ILP *formulation*, and has the form

$$\begin{aligned} \max \quad & c^T x && && (2.1)\\ \text{subject to} \quad & Ax \leq b && && (2.1.1)\\ & x_i \in \mathbb{Z} && \forall i \in I \ , && (2.1.2) \end{aligned}$$

where $I \subseteq \{1, \dots n\}$ is a subset of variable indices. In the case that all variables have to take integer values, we also call (2.1) a *pure integer linear program*. If the possible integer values are restricted to the set $\{0, 1\}$, we refer to (2.1) as a *zero-one formulation*. Solving a general integer linear program to optimality is an *NP*-complete problem as shown by Garey and Johnson (1979).

Many strategies that compute approximate or exact solutions for an ILP formulation (2.1) rely on the *linear programming relaxation* of (2.1) which results from dropping the integrality constraints (2.1.2).

At this point, we list some special cases of linear programs that correspond to important and well-studied optimization problems in weighted directed graphs and for which polynomial-time algorithms exist.

Definition 2.5 (Minimum-cost flow problem). Let $D = (V, A)$ be a directed graph with capacities $u \in \mathbb{R}_+^A$ and costs $c \in \mathbb{R}^A$, and let $b \in \mathbb{R}^N$ be a supply vector with

$$\sum_{i \in N} b_i = 0 \ .$$

Such a digraph with additional information is also referred to as a *network*. The task is to find a minimum-cost flow in D, *i.e.*, an optimal solution of the following linear program

$$\begin{aligned} \min \quad & c^T x && && (2.2)\\ \text{subject to} \quad & \sum_{a=(i,j)} x_a - \sum_{a=(j,k)} x_a = b_j && \forall j \in N\\ & 0 \leq x_a \leq u_a && \forall a \in A \end{aligned}$$

The following two problems are special cases of the minimum-cost flow problem:

Definition 2.6 (Maximum s-t flow problem). Let $D = (V, A)$ be a directed graph with capacities $u \in \mathbb{R}^A_+$ and let s and t be two nodes in N. Find a maximum flow from s to t in D, *i.e.*, an optimal solution of the following linear program

$$\begin{aligned} \min \quad & \mathbb{1}^T x && && (2.3)\\ \text{subject to} \quad & \sum_{a=(i,j)} x_a - \sum_{a=(j,k)} x_a = 0 && \forall j \in N \setminus \{s,t\} \\ & 0 \le x_a \le u_a && \forall a \in A \end{aligned}$$

Definition 2.7 (Shortest s-t path problem). Given a directed graph $D = (N, A)$ with arc weights l and two nodes $s, t \in N$, find a shortest path (with respect to l) from s to t. An equivalent linear program is:

$$\begin{aligned} \min \quad & l^T x && && (2.4)\\ \text{subject to} \quad & \sum_{a=(i,j)} x_a - \sum_{a=(j,k)} x_a = b_j && \forall j \in N \\ & 0 \le x_a && \forall a \in A \ , \end{aligned}$$

where $b_s = 1$, $b_t = -1$ and $b_i = 0$ for all $i \in N \setminus \{s,t\}$.

The dual linear program of (2.4) states the shortest path optimality conditions, see, *e.g.*, (Cook, Cunningham, Pulleyblank, and Schrijver, 1998),

$$\begin{aligned} \min \quad & x_t - x_s && && (2.5)\\ \text{subject to} \quad & x_j \le x_i + l_a && \forall a = (i,j) \in A \end{aligned}$$

2.4 Combinatorial Optimization

In general, a *combinatorial optimization problem* asks for a best solution from a finite set $\mathcal{I}$ of *feasible solutions* with respect to an *objective function* $f : \mathcal{I} \to \mathbb{R}$, *i.e.*, an element $I^* \in \mathcal{I}$ with $f(I^*) = \max\{f(I) \mid I \in \mathcal{I}\}$.

In this thesis, we will only consider linear objective functions and feasible solutions that correspond to subsets of a finite ground set. The following is a precise definition of this special case.

Definition 2.8 (Linear combinatorial optimization problem). Given a finite ground set E, a set $\mathcal{I} \subseteq \mathcal{P}(E)$ of feasible solutions and a vector $c \in \mathbb{R}^E$ of objective function coefficients, find a set $I^* \in \mathcal{I}$ with

$$\sum_{e \in I^*} c_e = \max \left\{ \sum_{e \in I} c_e \mid I \in \mathcal{I} \right\} \ .$$

The relation between linear combinatorial optimization problems and (integer) linear programming is as follows. We can characterize the feasible solutions by means of an

incidence vector x. Every feasible solution $F \subseteq E$ defines an element $x^F \in \mathbb{R}^E$ in the following manner:

$$x_e^F = \begin{cases} 1 & e \in F \\ 0 & e \notin F \end{cases} .$$

Every linear combinatorial optimization problem then corresponds to a polytope

$$P_{\mathcal{I}} = \text{conv}\{x^I \mid I \in \mathcal{I}\}$$

that is defined by the convex hull of the incidence vectors of feasible solutions. Now the combinatorial optimization problem can be written as

$$\max\{c^T x \mid x \in P_{\mathcal{I}}\} \quad . \tag{2.6}$$

The *linear description* is an equivalent description of the polytope $P_{\mathcal{I}}$ in terms of a linear program, see, *e.g.*, (Schrijver, 1986):

$$\max\{c^T x \mid Ax \leq b\} \quad . \tag{2.7}$$

Let $a \in \mathbb{R}^E \setminus \{0\}$ be a vector. An inequality $a^T x \leq a_0$, where $a_0 \in \mathbb{R}$ is a scalar, is *valid* with respect to the polytope $P_{\mathcal{I}}$ if $P_{\mathcal{I}} \subseteq \{x \in \mathbb{R}^E \mid a^T x \leq a_0\}$. We can optimize a linear objective function over a polytope in polynomial time if and only if we can solve the *separation problem* in polynomial time (Grötschel, Lovász, and Schrijver, 1994; Padberg and Rao, 1981; Karp and Papadimitriou, 1982). The separation problem can be stated as follows: Given a vector $\bar{x} \in P_{\mathcal{I}}$, find a vector $a \in \mathbb{R}^E$ and a scalar $a_0 \in \mathbb{R}$ so that $a^T \bar{x} \leq a_0$ is a violated valid inequality with respect to $P_{\mathcal{I}}$, that is $a^T \bar{x} > a_0$, or prove that no such pair (a, a_0) exists.

Unfortunately, for most of the interesting combinatorial optimization problems, only a small fraction of the linear description is known. For *NP*-hard optimization problems, the full linear description cannot be found, unless *NP* is equal to co-*NP* (Karp and Papadimitriou, 1982). However, in many cases it is practicable to find a linear program with additional integrality constraints, *i.e.*, an integer linear program, to give a characterization of the integral points within the polytope.

In the following, we present generic algorithmic approaches to solve linear combinatorial optimization problems that exploit the equivalence of the original problem and an appropriate integer linear program. In this thesis, we will develop several zero-one integer linear programs, and we will explain the generic algorithms within the context of zero-one problems.

Linear programming-based *branch-and-bound* is a divide-and-conquer technique that solves the original problem by splitting it recursively into smaller subproblems, resulting in a tree of subproblems. For every subproblem, a branch-and-bound algorithm computes *local upper bounds* and tries to improve a *global lower bound*.[1] Initially, the global lower bound is usually initialized by a heuristics.

The root of the enumeration tree corresponds to the original problem, and nodes that are not leaves of the tree have exactly two children. One of these children corresponds to

[1] Note that in a minimization problem, the lower bounds are local, and the upper bound is global.

the subproblem in which one binary variable (the *branching variable*) is set to zero, and the other one represents the problem in which the branching variable is equal to one. In many cases, a branch-and-bound algorithm is as successful as the computation of upper bounds at the inner nodes of the enumeration tree. At each node, a *local upper bound* u_{loc} can be computed by solving the LP-relaxation for the corresponding subproblem.

Three cases arise regarding the relation of the upper bound and lower bounds.

1. If the solution of the relaxation happens to be feasible for the original problem and its objective function value exceeds the current global lower bound l_{glob}, the algorithm updates the lower bound and memorizes the solution. Additionally, the current node is *fathomed*, since no optimal solution of subproblems corresponding to descendants of that node can be better than the current solution.

2. In case the local upper bound is lower than, or equal to the global lower bound, the algorithm fathoms the current node of the tree for the same reasons as above.

3. A third case arises if the local upper bound exceeds the global lower bound and the optimal solution of the LP-relaxation is not feasible for the original problem. In this case, the algorithm performs a branching step by selecting a binary branching variable and adds the resulting two problems in which the chosen variable is fixed to zero and one, respectively, to the list of subproblems.

In addition to determining the local upper bounds, the fractional solutions of LP-relaxations can also be used to improve the global lower bound. Problem-specific heuristics may exploit the information given in the fractional solution for the relaxation, *e.g.* by rounding or by setting variables according to a certain threshold.

At the beginning of the computation the list of subproblems contains only the original problem. The algorithm repeatedly selects an open subproblem from this list, computes the lower and upper bounds and, depending on the bounds, performs one of the above three operations. If the list of subproblems is empty, the memorized solution that corresponds to the global lower bound is the optimal solution.

Algorithm 2.1 summarizes the steps of the generic branch-and-bound algorithm.

Branch-and-cut algorithms extend LP-based branch-and-bound techniques by combining them with the *cutting plane approach*. We will only give a sketch of the algorithm, a detailed description can be found, *e.g.*, in (Jünger, Reinelt, and Thienel, 1995).

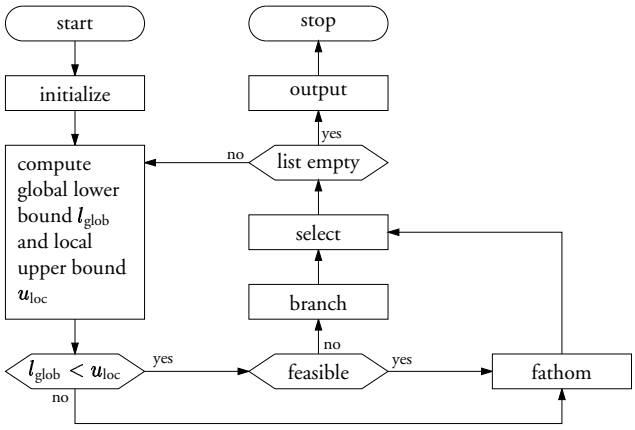
We start with a description of the cutting plane approach. Suppose the description of a combinatorial optimization problem as a zero-one integer linear programming problem contains a potentially exponential number of inequalities. A cutting plane approach starts with an LP-relaxation that arises from considering only a small subset of the inequalities and computes an optimal solution for the corresponding linear program. In a following step, the algorithm tries to find violated inequalities, *e.g.*, by considering the separation problem for certain classes of inequalities. If no violated constraint exists, the solution is also optimal for the original problem. Otherwise, the new inequalities cut off the solution for the relaxation. Algorithm 2.2 illustrates a simple cutting plane approach.

Depending on the hardness of the separation problem, either exact or heuristic algorithms may identify violated inequalities. Note that in the case of a heuristic strategy

Algorithm 2.1 *Generic branch-and-bound algorithm*

Input: Integer linear program
ILP = $\max\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_i \in \{0,1\} \text{ for all } i \in I\}$

Output: Optimal solution for ILP



violated constraints may exist that are not found by the separation algorithm. However, Gomory (1958) shows that a general cutting plane based on rounding can always be found. Hence, Algorithm 2.2 terminates after a finite number of steps.

Algorithm 2.2 *Generic cutting plane algorithm*

Input: Integer linear program
ILP = $\max\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_i \in \{0,1\} \text{ for all } i \in I\}$

Output: Optimal solution for ILP

1: (A', b') = small subset of (A, b);
2: **repeat**
3: solve the linear program $c^T \bar{x} = \max\{c^T x \mid A'x \leq b', x \in \mathbb{R}^n\}$;
4: **if** $\bar{x}$ not feasible for ILP **then**
5: generate cutting plane $(a, a_0), a \in \mathbb{R}^n, a_0 \in \mathbb{R}$ with $a^T \bar{x} > a_0$ and $a^T y \leq a_0$ for all $y \in \{x \mid Ax \leq b, x_i \in \{0,1\} \text{ for all } i \in I\}$;
6: add $a^T x \leq a_0$ to (A', b');
7: **until** $\bar{x}$ is feasible solution of ILP

Branch-and-cut algorithms search for cutting planes at every node of the branch-and-bound enumeration tree, thus improving the bound of the LP-relaxation. For many linear optimization problems extending the branch-and-bound framework by additional problem-specific and tight cutting planes decreases the number of nodes needed in the enumeration tree significantly and leads to highly successful algorithms in practice.

Chapter 3

Constraint Graphs

con'straint *s.* **1.** Zwang *m*, Nötigung *f*: **under** ~ unter Zwang, zwangsweise; **2.** Beschränkung *f*; **3.** a) Befangenheit *f*, b) Gezwungenheit *f*; **4.** Zurückhaltung *f*.
Langenscheidts Handwörterbuch Englisch.

Throughout this thesis constraint graphs play an important role. This chapter defines basic concepts and introduces several properties that are important for this work. Informally, directed edges of a constraint graph represent precedence constraints between objects corresponding to their source and target nodes. Weights on the arcs may additionally specify these relations. Constraint graphs occur in many areas; among them are artificial intelligence (constraint satisfaction problems, temporal and spatial reasoning), operations research (scheduling, systems of difference constraints), and VLSI design. In scheduling, for example, the nodes of a constraint graph correspond to the jobs and the weighted edges characterize the temporal constraints between these jobs. In the area of VLSI design, automatic design tools use constraint graphs to represent the placement constraints between the components that make up a chip layout. In this thesis, we will use pairs of constraint graphs with a similar meaning as in symbolic chip layout. All of these areas use different notations and identical or similar results have been developed in different contexts. This chapter summarizes these results and presents constraint graphs in a way suitable for further usage in orthogonal placement problems.

It turns out to be a crucial problem to compute so-called *assignments* for constraint graphs, *i.e.*, values for the nodes of a constraint graph that respect the precedence constraints coded in the arcs. In the following chapters on compaction and map labeling these assignments will play an important part in setting the coordinates of the orthogonal objects: in two-dimensional compaction problems in graph drawing, assignments determine the coordinates of vertices and bends, in map labeling problems, they fix the positions of the labels.

Two-dimensional compaction problems in orthogonal graph drawing and map labeling problems have several interesting characteristics in common: First, both problem types ask for a placement of iso-oriented objects, *i.e.*, their solutions are fully characterized by a set of line segments that are parallel to the axes of the Cartesian coordinate system. Therefore, it suffices to specify the coordinates of these line segments. Second, we already know some placement relations in advance—in a compaction problem this information stems from the shape of the drawing, whereas in a labeling problem the labels have to be placed in certain, predefined regions. A third common feature of both problems types is that they decompose naturally into two largely independent subproblems: Since the underlying ob-

jects are iso-oriented, this results in a horizontal and a vertical problem component.

Our new approach to these types of orthogonal placement problems is based on a pair of constraint graphs—one for each problem component. Nodes in the *horizontal constraint graph* D_x correspond to x-coordinates of problem-specific objects. Weighted directed edges represent horizontal distance relations between the objects that correspond to the respective endpoints. Similarly, the directed graph D_y codes the vertical relationships. The key idea is to treat both components separately to the greatest possible extent and to link them with additional constraints at their points of contact. In this work, constraint graphs serve the purpose to transfer the originally geometric problem formulations into a combinatorial environment: Both the compaction and the map labeling problems will be reformulated as combinatorial problems in a pair of constraint graphs.

Definition 3.1 (Constraint graph). A *constraint graph* is a weighted directed graph $D = (N, A, w)$ with arc weights $w : A \to \mathbb{Q}$. The graph is associated with an *assignment* $c : N \to \mathbb{Q}$. An arc $a = (i, j) \in A$ corresponds to the inequality

$$c(j) - c(i) \geq w(a) \quad . \tag{3.1}$$

Definition 3.2 (Types of assignments). An assignment $c : N \to \mathbb{Q}$ is *feasible* for a constraint graph $D = (N, A, w)$ if it satisfies all constraints encoded in the weighted arcs set A. The *span* of an assignment c is its maximal minus its minimal value, *i.e.*,

$$\mathrm{sp}(c) = \max_{n \in N} c(n) - \min_{n \in N} c(n) \quad .$$

The *distance* of an arc $a = (i, j) \in A$ is

$$\Delta(a) = c(j) - c(i) \quad ,$$

and the *distance of an assignment* is the sum of the distances of its arcs, *i.e.*,

$$\Delta(c) = \sum_{a=(i,j) \in A} \Delta(a) = \sum_{(i,j) \in A} c(j) - c(i) \quad .$$

A feasible assignment c is

$$\begin{cases} \textit{minimal} & \text{if its span } \mathrm{sp}(c) \text{ is minimal} \\ \textit{minimax} & \text{if the largest distance } \max_{a \in A} \Delta(a) \text{ is minimal} \\ \Delta\textit{-minimal} & \text{if its distance } \Delta(c) \text{ is minimal} \end{cases}$$

among all feasible assignments. An assignment *respects* an arc set B if it is feasible for the subgraph induced by B.

Observation 3.1. *A feasible assignment c for a constraint graph D is feasible for every subgraph of D.*

Proof. Assume c is not feasible for a subgraph D_s of D. Then there exists an arc $a = (i, j)$ in D_s with $c_j - c_i < w_a$. Since all arcs of D_s are also contained in the arc set of the supergraph, the same situation occurs in D, resulting in the contradiction that c is not feasible for D. Informally speaking, a solution which respects a set of constraints will also satisfy each subset of this constraint set. ∎

Constraint graphs have a strong connection to minimum-cost flow problems and in particular to shortest path problems. The assignment function c of Definition 3.1 is closely related to the *node potentials* in network flow theory (see, *e.g.*, Ahuja, Magnanti, and Orlin, 1993)[1]: Substituting $d(i) = -c(i)$ for all $i \in N$ and $l(a) = -w(a)$ in (3.1) results in

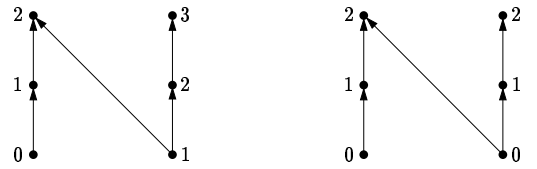
$$d(j) \leq d(i) + l(a) \ ,$$

the optimality condition for shortest path distances d. This equivalence will be exploited in the computation of feasible assignments in Sections 3.1- 3.3.

It is easy to see that a minimal assignment may not be Δ-minimal. But also Δ-minimality, although seemingly the stronger criterion, does not imply minimality:

Lemma 3.1. *A Δ-minimal assignment is not necessarily minimal.*

Proof. Consider the constraint graph in Figure 3.1 and the two assignments in (a) and (b). The distance $\sum_{(i,j)\in A} c(j) - c(i)$ is 5 in the Δ-minimal assignment (left) and 6 in the minimal assignment (right), so the assignment in Figure 3.1(b) is not Δ-minimal. The span, however, is 3 in the left assignment and only 2 in the right assignment, so the assignment in Figure 3.1(a) cannot be minimal. ∎



(a) Δ-minimal assignment (b) Minimal assignment

Figure 3.1: Two assignments, $w(a) = 1$ for all arcs a

We will often deal with paths in constraint graphs. Summing up the individual inequalities for each arc in a path leads to the following observation.

Observation 3.2. *Let $D = (N, A, w)$ be a constraint graph, let c be a feasible assignment for D, and let $p = \big((n_1, n_2), (n_2, n_3), \ldots, (n_{k-1}, n_k)\big)$ be a path in A. Then the following inequality holds.*

$$c(n_k) - c(n_1) \geq \sum_{i=1}^{k-1} w\big((n_i, n_{i+1})\big) \ .$$

Computing feasible, minimal, minimax, or Δ-minimal assignments for constraint graphs is the core of our new approaches to compaction and map labeling problems which we will present in Chapters 4 and 5. The following well-known theorem characterizes circumstances under which a constraint graph admits a feasible assignment.

[1] Often, even c is referred to as a potential in the literature. We stick to the term *assignment*, however, to avoid confusion with the potentials in the related shortest path problems.

Theorem 3.1 (*e.g.*, Cook *et al.* (1998)). *Given a constraint graph $D = (N, A, w)$. A feasible assignment $c : N \to \mathbb{Q}$ exists if and only if A does not contain a directed cycle of positive weight.*

Theorem 3.1 characterizes in which cases feasible assignments exist but does not state how to generate them. The following sections contain algorithms that compute minimal, Δ-minimal, and minimax assignments. We consider various special cases that allow for more efficient algorithmic solutions.

3.1 Computing Minimal Assignments

Let $D = (N, A, w)$ be a constraint graph. In this section we demonstrate that finding a minimal assignment c in D corresponds to solving a shortest path problem in a related graph. The interpretation of c in the original graph corresponds to *longest paths*. Whereas the problem of computing longest paths in a directed graph with general arc weights is *NP*-complete, (see, *e.g.*, Garey and Johnson, 1979), it can be solved efficiently in the absence of positive cycles. The resulting methods which we will present in the following are also referred to as *longest path methods*. In operations research, the relation between assignments and longest paths in the constraint graphs is known as the *min-potential max-work theorem*, see, *e.g.*, (Grötschel *et al.*, 1994).

In order to access the minimal and maximal values in c we introduce two new nodes in N; a super source s and a super sink t. We require

$$c(s) \leq c(i) \leq c(t) \qquad \text{for all } i \in N \ .$$

The corresponding arcs that have to be added to A are

$$A_s = \{(s, i) \mid i \in N\} \quad \text{and} \quad A_t = \{(i, t) \mid i \in N\}$$

of weight

$$w(a) = 0 \quad \text{for all } a \in A_s \cup A_t \ .$$

The computation of a minimal assignment c then corresponds to solving the following linear program:

$$\begin{array}{rll} \min & c(t) - c(s) & \qquad \text{(LP.1)} \\ \text{subject to} & c(j) - c(i) \geq w(a) & \forall a = (i, j) \in A \\ & c(i) - c(s) \geq 0 & \forall i \in N \\ & c(t) - c(i) \geq 0 & \forall i \in N \end{array}$$

We substitute $l(a) = -w(a)$ for all $a \in A \cup A_s \cup A_t$ and $d(i) = -c(i)$ for all $i \in N \cup \{s, t\}$ and rewrite problem (LP.1) as

$$\begin{array}{rll} \max & d(t) - d(s) & \qquad \text{(LP.2)} \\ \text{subject to} & d(j) \leq d(i) + l(a) & \forall a = (i, j) \in A \cup A_s \cup A_t \end{array}$$

Problem (LP.2) is dual to a shortest path problem (see Definition 2.7 on page 23) and can be solved with the Bellman-Ford algorithm. Algorithm 3.1 on page 31 summarizes the steps to compute a minimal assignment, and Figure 3.2 shows the sequence of steps with an example. The first two pictures show the constraint graph in which the assignment should be computed before and after insertion of A_s and A_t. Figure 3.2(c) shows the shortest path tree, Figure 3.2(d) displays the initial constraint graph together with the minimal assignment.

Algorithm 3.1 *Bellman-Ford assignment*

Input: Constraint graph $D = (N, A, w)$

Output: Minimal assignment for D or detect positive cycle

1: introduce super source s and super sink t;
2: add the arcs in A_s and A_t;
3: **for all** $a \in A \cup A_s \cup A_t$ **do** $l(a) = -w(a)$;
4: compute shortest s-t path in D with lengths l using the Bellman-Ford algorithm, store the distances in d;
5: **if** Bellman-Ford detects negative cycle **then**
6: output "positive cycle";
7: **else**
8: delete s, t and the incident arcs A_s and A_t;
9: **for all** $i \in N$ **do** $c(i) = -d(i)$;

Theorem 3.2. *Algorithm Bellman-Ford assignment computes a minimal assignment for a constraint graph $D = (N, A, w)$ or reports a positive cycle in time $O(|N| \cdot |A|)$.*

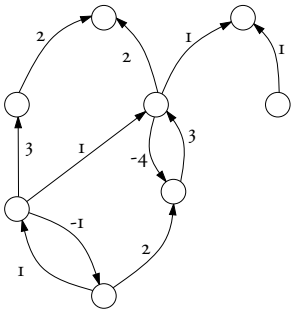
Proof. For the proof of correctness note that the constraints that correspond to the additional arcs in A_s and A_t do not interfere with the original constraints. Furthermore, the corresponding arcs do not induce cycles in the graph since $A_s \cup A_t$ is a set of disjunct s-i-t-paths for all nodes $i \in N$.

The above transformation from (LP.1) to (LP.2) shows the correspondence between the negative distance values in the shortest path problem and the values for the assignment. The correctness of Theorem 3.2 follows from the correctness of the Bellman-Ford algorithm for computing shortest paths (Bellman, 1958; Ford and Fulkerson, 1962; Moore, 1959).

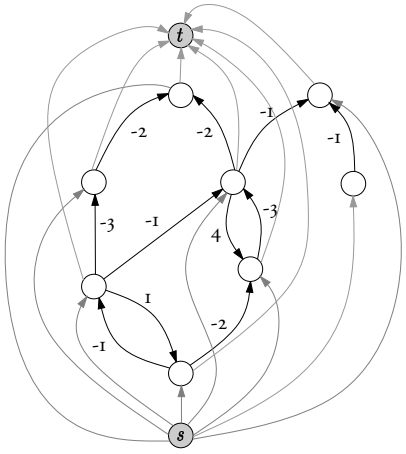
The running time is clearly dominated by the time needed for the shortest path computation that is $O(|N| \cdot |A|)$. Steps (2), (3) and (8) take time $O(|A|)$, step (9) takes time $O(|N|)$ and the other steps can be done in constant time. ∎

The following remarks consider two special cases that lead to more efficient algorithms for the computation of a minimal assignment.

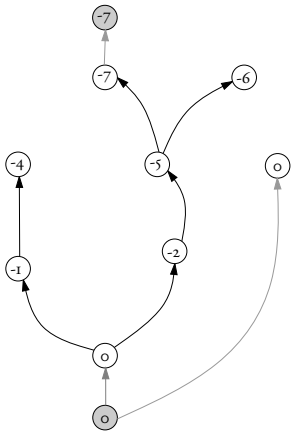
Remark 3.1. In case a feasible assignment is known in advance, an observation by Maley (1987) allows us to use Dijkstra's shortest path algorithm in line (4) of Algorithm 3.1



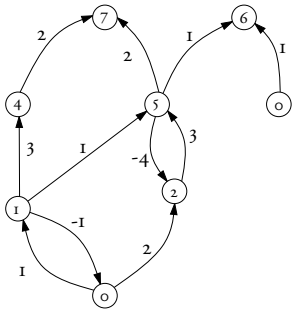
(a) Constraint graph D; arcs labeled with weights



(b) After execution of line 3; arcs labeled with lengths, unlabeled arcs have length zero



(c) Shortest path tree after execution of line 4; nodes labeled with d



(d) After execution of line 9; constraint graph D, nodes labeled with assignment c

Figure 3.2: Computing a minimal assignment by solving a shortest path problem from super source s to super sink t

(Dijkstra, 1959). Changing the weight to

$$w'(a) = w(a) + c(j) - c(i) \qquad \forall a = (i,j) \in A \tag{3.2}$$

results in a constraint graph $D = (N, A, w')$ with non-positive weights. The transformation (3.2) preserves the shortest paths. Since the lengths in the shortest path computation are equal to the negative weights, we can use Dijkstra's algorithm; the running time of Algorithm 3.1 is then $O(|A| + |N| \log |N|)$.

Remark 3.2. Lengauer and Mehlhorn (1986) present a linear time algorithm in case the constraint graph is a so-called *chain-dag*, *i.e.*, if it consists of disjoint chains of antiparallel arcs.

We now consider the important special case in which all arc weights are non-negative. If a constraint graph D satisfies this property, then each cycle in D has positive weight. The absence of arcs with negative weight enables us to use a more efficient algorithm to compute a minimal assignment—we exploit the fact that shortest paths can be computed considerably faster in acyclic graphs. In operations research, the longest path in an acyclic constraint graph is also known as the *critical path*, since its length determines the span of the assignment. In project scheduling, for example, the span corresponds to the minimum project duration. Algorithms for this problems are also referred to as *critical-path methods*.

Algorithm 3.2 *Topological Order assignment*

Input: Constraint graph $D = (N, A, w)$ with $w(a) \geq 0$ for all $a \in A$

Output: Minimal assignment for D or detect positive cycle

1: introduce super source s and super sink t;
2: add the arcs in A_s and A_t;
3: **for all** $a \in A \cup A_s \cup A_t$ **do** $l(a) = -w(a)$;
4: compute shortest s-t path in D with lengths l using topological order, store the distances in d;
5: **if** the topological order algorithm detects cycle **then**
6: output "positive cycle";
7: **else**
8: delete s, t and the incident arcs A_s and A_t;
9: **for all** $i \in N$ **do** $c(i) = -d(i)$;

Theorem 3.3. *Algorithm Topological Order assignment computes a minimal assignment for a constraint graph $D = (N, A, w)$ with $w(a) \geq 0$ for all $a \in A$ or reports a positive cycle in time $O(|N| + |A|)$.*

Proof. The correctness follows by similar considerations as demonstrated in the proof of Theorem 3.2 and by the correctness of topological sort (Knuth, 1968). As for Algorithm 3.1, the running time is dominated by the time needed to compute the shortest paths or to detect a cycle. In the case of non-negative arc weights the time is $O(|N| + |A|)$. ∎

3.2 Computing Δ-Minimal Assignments

The previous section demonstrates the computation of minimal assignments by solving a shortest path problem. In this section, we consider the computation of Δ-minimal assignments. We will present an algorithm that constructs such an assignment by solving a dual network flow problem and by obtaining the primal solution through a computation of shortest paths in the residual network.

Like in the case of minimal assignments we take a closer look at the linear program that corresponds to the problem of finding a Δ-minimal assignment:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c(j) - c(i) && && \text{(LP.3)} \\ \text{subject to} \quad & c(j) - c(i) \geq w(a) && \forall a = (i,j) \in A \ . \end{aligned}$$

We rewrite (LP.3) by rearranging the objective function by means of the net degree as defined on page 17:

$$\begin{aligned} \min \quad & \sum_{n\in N} \delta_{\text{net}}(n) \cdot c(n) && && \text{(LP.4)} \\ \text{subject to} \quad & c(j) - c(i) \geq w(a) && \forall a = (i,j) \in A \ . \end{aligned}$$

The dual of (LP.4) is

$$\begin{aligned} \max \quad & \sum_{a\in A} w(a) \cdot y(a) && && \text{(LP.5)} \\ \text{subject to} \quad & \sum_{(j,i)\in A} y\big((j,i)\big) - \sum_{(i,j)\in A} y\big((i,j)\big) = \delta_{\text{net}}(i) && \forall i \in N \\ & y(a) \geq 0 && \forall a \in A \ . \end{aligned}$$

By rearranging (LP.5) to

$$\begin{aligned} \min \quad & \sum_{a\in A} -w(a) \cdot y(a) && && \text{(LP.6)} \\ \text{subject to} \quad & \sum_{(i,j)\in A} y\big((i,j)\big) - \sum_{(j,i)\in A} y\big((j,i)\big) = -\delta_{\text{net}}(i) && \forall i \in N \\ & y(a) \geq 0 && \forall a \in A \ . \end{aligned}$$

it is in the standard form of a minimum-cost flow problem, see Definition 2.5 on page 22. Note that supplies and demands sum up to zero, *i.e.*, $\sum_{n\in N} -\delta_{\text{net}}(n) = 0$, and that we may choose global capacities that are equal to the sum of all supplies, see, *e.g.*, (Ahuja *et al.*, 1993). Moreover, the underlying network of (LP.6) is the original constraint graph D, the costs of the arcs are their negative weights—similar as in the previous section—and the supplies and demands of the nodes are equal to their negative net degree.

Solving this minimum-cost flow problem leads to a dual solution that consists of values $y(a)$ for each arc a in D. Employing duality theory for minimum-cost flow problems allows to reduce this problem again to a shortest path problem; to get a primal solution, we have to compute shortest paths in the residual network (see, *e.g.*, Ahuja *et al.*, 1993).

Algorithm 3.3 *Dual minimum-cost flow assignment*

Input: Constraint graph $D = (N, A, w)$

Output: Δ-minimal assignment for D or detect positive cycle

1: **for all** $n \in N$ **do** $b(n) = -\delta_{\text{net}}(n)$;
2: **for all** $a \in A$ **do** $c(a) = -w(a)$;
3: compute minimum-cost flow x in D with supply vector b and costs c;
4: **if** the minimum-cost flow algorithm detects a negative-cost cycle **then**
5: output "positive cycle";
6: **else**
7: construct residual network R w.r.t. the flow x; // the node set of R is N
8: s = arbitrary node in N;
9: compute shortest paths from s in R w.r.t. lengths that are equal to the costs in N_x, store the distances in d;
10: **for all** $i \in N$ **do** $c(i) = -d(i)$;

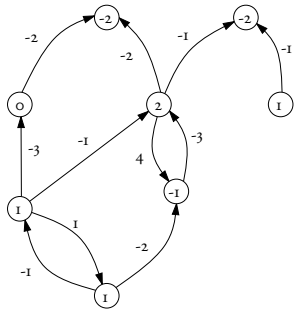
Theorem 3.4. *Algorithm Dual Minimum-Cost Flow assignment computes a Δ-minimal assignment in a constraint graph (N, A, w) or reports a positive cycle in time*

$$O\big((|A| \log |N|)(|A| + |N| \log |N|)\big) \ .$$

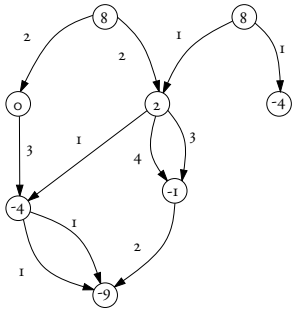
Proof. The correctness follows by the transformation from (LP.3) to (LP.6), by the correctness of the minimum cost flow algorithms, and by the shortest path optimality conditions, see, *e.g.* (Ahuja *et al.*, 1993). The running time of Algorithm 3.3 is dominated by the the computation of a minimum cost flow in line (3). The *capacity scaling algorithm* due to Edmonds and Karp (1972) accomplishes this task in time $O\big((|A| \log U)(|A| + |N| \log |N|)\big)$, where U is an upper bound for the largest supply or demand. In Algorithm 3.3, the supplies and demands depend on the net degree, and it holds $U = O(|N|)$. ∎

Remark 3.3. In case the cost of an optimal solution of the minimum-cost flow problem is low, the algorithm by (Garg and Tamassia, 1997) results in a better running time. Let χ be the cost of a minimum-cost flow. Line (3) in Algorithm 3.3 can be executed in time $O(\chi^{3/4}|A|\sqrt{\log |N|})$.

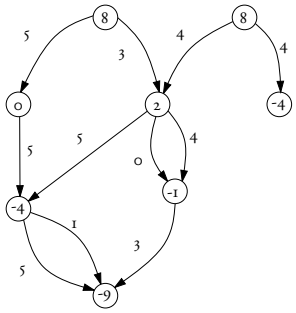
Figure 3.3 illustrates the computation of a Δ-minimal assignment with Algorithm 3.3. The constraint graph in the figure is the one from Figure 3.2(a). The first picture, Figure 3.3(a), shows the network of the dual problem. Since this network has negative costs, we transform it into an equivalent network, displayed in Figure 3.3(b). All capacities can be set to the total supply of the nodes. The next two pictures show the minimum cost flow in the transformed and the original network, respectively. Based on the latter we construct the residual network displayed in Figure 3.3(e). Negative shortest path distance values in this network constitute the Δ-minimal assignment. For a more detailed explanation of the transformations and the concept of residual networks, see (Ahuja *et al.*, 1993).



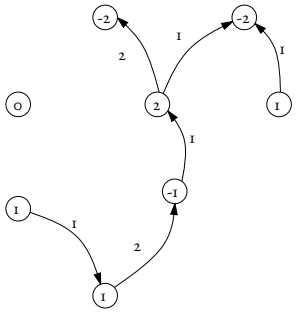
(a) Dual network of constraint graph D in Figure 3.2(a); arcs labeled with costs, nodes labeled with supply/demand



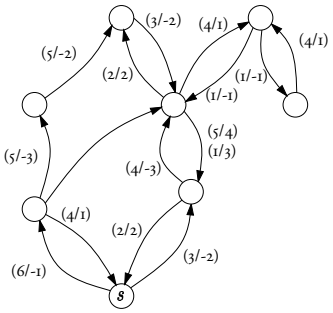
(b) The transformed network. All capacities are equal to five, arcs labeled with costs



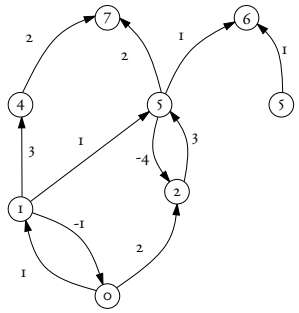
(c) Minimum-cost flow in transformed network



(d) Minimum-cost flow in the network displayed in (a). Arcs with non-zero flow labeled with flow values



(e) Residual network with respect to the flow. Arcs labeled with (capacity/cost)



(f) Original constraint graph with Δ-minimal assignment that is equal to the negative shortest path distances in the residual network

Figure 3.3: Computing a Δ-minimal assignment by solving a dual minimum-cost flow problem

Theorem 3.5. *Let $D = (N, A, w)$ be a constraint graph. If all weights are integral, then minimal and Δ-minimal assignments with integral values exist.*

Proof. The result follows from the integrality property of minimum-cost flow problems, see, *e.g.* (Ahuja *et al.*, 1993, Theorem 9.10). The same holds for shortest paths problems since they are special flow problems. Furthermore, algorithms exist that compute integral assignments. Throughout this thesis, we will assume that Algorithms 3.1 and 3.3 compute integral assignments for integral input data. ∎

Remark 3.4. In the next two chapters we introduce a new approach to solve orthogonal placement problems by using a pair of constraint graphs. Assignments for such a pair of graphs will determine the coordinates of a graph drawing or the placement of labels.

There is, however, another application of constraint graphs in the field of graph drawing, the algorithm by Sugiyama, Tagawa, and Toda (1981). This method for drawing hierarchical graphs works in three phases. The first phase of the algorithm, also referred to as *rank assignment*, partitions the vertex set into subsets called layers. The layers determine the y-coordinate of the vertices in a drawing. Similar as in orthogonal graph drawing, one optimization goal of this phase is to keep the edges short—the problem in the rank assignment phase is equal to a one-dimensional compaction problem.

Gansner, Koutsofios, North, and Vo (1993) mention that an optimal rank assignment (which corresponds to a Δ-minimal assignment in the corresponding constraint graph) can be computed via minimum-cost flow but describe a rather complicated network-simplex method to solve the problem. The rank assignment problem in hierarchical graph drawing can be solved optimally by applying Algorithm 3.3.

3.3 Computing Minimax-Assignments

Let $c_{\max}$ be a variable that corresponds to largest distance of an arc in a constraint graph $D = (N, A, w)$. We express the problem to find a minimax assignment by the following linear program.

$$\begin{array}{rll} \min & c_{\max} & \\ \text{subject to} & c(j) - c(i) \geq w(a) & \forall a = (i,j) \in A \\ & c(j) - c(i) \leq c_{\max} & \forall (i,j) \in A \end{array} \tag{LP.7}$$

The linear program (LP.7) can be solved in polynomial time and determines a minimax assignment for the constraint graph D.

Chapter 4

Compaction in Graph Drawing

kompakt [*lat.-fr.*]: 1. (ugs.) massig, gedrungen. 2. undurchdringlich, dicht, fest. 3. gedrängt, kurzgefasst, das Wesentliche zusammenfügend.

Duden. Fremdwörterbuch.

In the area of graph drawing, the term *compaction* denotes the process of transforming a given orthogonal representation into an actual drawing, *i.e.*, the last phase within the topology-shape-metrics scheme. The goal is to produce an orthogonal grid embedding with short edges that does not require too much area; the most common optimization criteria are width, height, half-perimeter, area, and total and maximum edge length of the resulting drawing. In the final phase of the topology-shape-metrics scheme in graph drawing, minimizing total edge length and area are the primary goals. Figure 4.1 shows that different drawings of the same orthogonal representation do not necessarily have to be very similar.

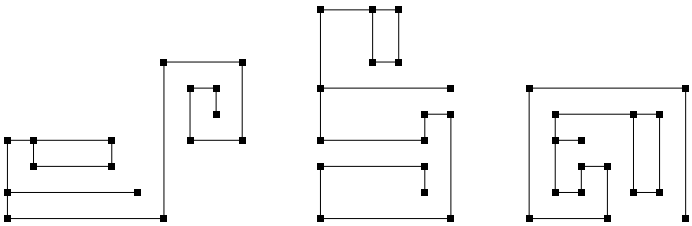


Figure 4.1: Three different drawings of the same orthogonal representation

Traditionally, due to its origin in the area of VLSI design, compaction also refers to changing an existing layout in order to improve one or more of the above criteria. VLSI circuits can contain more than a million elements which are organized into subcircuits, the so-called components. Layout problems emerging in chip design deal with positioning components and their interconnecting wires according to a set of design rules and mostly ask for a placement of minimum area.

This chapter contains a common formal framework for compaction problems that is based on a pair of constraint graphs. First, in Section 4.1, we formally introduce the two-dimensional compaction problems we deal with in this chapter and present corresponding complexity results.

The core of this chapter is Section 4.2 where we develop a combinatorial characterization of the compaction problems: We introduce two special constraint graphs—one for

each direction of the compaction—and show that if this pair of graphs satisfies the path- and cycle-based combinatorial property of *completeness* we are able to solve the two-dimensional compaction problems in polynomial time. We present previously known compaction heuristics that are based on polynomial-time compactible classes of orthogonal representations in the light of the new characterization. We distinguish between *constructive heuristics* that produce a feasible drawing for a given orthogonal representation and *improvement heuristics* that transform existing solutions in order to achieve better solutions. In particular, we take a closer look at the one-dimensional compaction scheme and study properties of algorithms within that scheme. We conclude the section by deriving combinatorial reformulations of the two-dimensional compaction problems.

Based on these new combinatorial problem versions, we develop exact polynomial-time compaction algorithms in Section 4.3. We present both branch-and-bound and branch-and-cut algorithms that are based on integer linear programming formulations for the new combinatorial problems. We also discuss related exact techniques from the area of VLSI design and compare the approaches.

Section 4.4 contains a comprehensive computational study and comparison of different compaction algorithms. The chapter ends with the conclusions we draw from our experimental results.

4.1 Compaction Problems

The compaction problems we consider in this chapter emerge in the area of graph drawing and are similar, but not equal, to compaction problems in VLSI design. This section provides formal definitions of the compaction problems as they occur within the topology-shape-metrics approach to orthogonal graph drawing. As described in the previous chapter, algorithms that fit in this scheme produce in their second phase a simple orthogonal representation H that fixes the shape of an orthogonal drawing of a given four-planar graph G. The last step within the scheme is the compaction phase. It is concerned with assigning coordinates to the vertices in H which may correspond to vertices in G or to bends along the edges of G. At this point the topology as well as the orthogonal shape of the drawing are fixed.

This is one of the issues distinguishing the compaction problems of this chapter from those in VLSI design. The fabrication of a VLSI chip is often modeled in two phases: A first step deals with the placement of the components; here two of the most common goals are to minimize the estimated area or total wire length. The second step is concerned with finding the paths for the wires. Each wire connects two specified positions of the components. The two steps are usually referred to as the *placement phase* and the *routing phase*. Compaction is a post-processing step in VLSI design. Often, a more detailed specification divides the layout problem into the phases component placement, global routing, topological compaction, detailed routing, and geometrical compaction.

The compaction problems in VLSI are more difficult than in graph drawing: Although VLSI compaction also denotes the process of transforming a topological description, known as *symbolic layout*, into an actual placement, the *physical mask layout*, it is at the same

time closely related to the fabrication technology. Many of the technological aspects, *e.g.*, electrical stability properties or interaction between the circuit elements do not translate easily into a combinatorial description.

A symbolic layout consists of graphic symbols that represent circuit elements commonly known as *features*, *e.g.*, transistors, contacts, and capacitators. *Design rules* model the relative placement of these features. The rules include shape and minimum size rules that describe the shape of components which in turn consist of several features. Additional separation rules between different components require minimum distances that have to be respected during the fabrication process.

The major difference between compaction problems in VLSI and in graph drawing is that a topological description given by the design rules needs not fully specify the embedding of the underlying graph. A symbolic layout may still allow the permutation of wires along the boundary of a component, whereas in an instance of the compaction problem within the topology-shape-metrics scheme the topology is fixed. A second difference between problems in the two research areas is that research in VLSI design concentrates on fast compaction strategies due to the large sizes of instances.

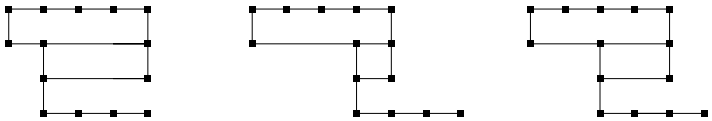
Common to both VLSI and graph drawing is that the compaction process has an influence on a variety of optimization goals. In the area of graph drawing, the compaction phase affects several of the aesthetic criteria that measure the quality of a drawing Γ:

- The area $A(\Gamma)$ of the smallest surrounding iso-oriented rectangle should be as small as possible. The more area a drawing uses, the more it has to be scaled down in order to fit on a computer screen or a sheet of paper (here, the aspect ratio $a(\Gamma)$ plays a role, too). Similar criteria are the width $W(\Gamma)$ and height $H(\Gamma)$ of the drawing, and their sum, the half-perimeter $P(\Gamma)$. Area minimization is the most important goal in VLSI design, since the area used by a chip has a large effect on its delay, *i.e.*, the time the electrical signals need to pass through the wires of the chip. Furthermore and more importantly, the area has a direct impact on the percentage of correctly working chips. This parameter is also called the *chip yield* and often depends exponentially on the area of the layout.

- Short edges allow a better understanding of a drawing, since they keep related objects close to each other. In VLSI, short wires improve the electrical qualities; their length affects signal delay. In many applications the minimization of total edge length $L(\Gamma)$ is the most desirable criterion. For certain problems it is more important that every single edge is not too long, which results in the problem of minimizing the maximum edge length $l_{\max}(\Gamma)$. *E.g.*, in models of wire delay, the dependence on the maximum edge length ranges from logarithmic to quadratic influence on the cost function. In certain applications, some edges are more important than others and should be kept short: This gives rise to a weighted total edge length minimization problem. Even a request for uniform edge lengths is conceivable; this would lead to the minimization of the deviations in edge length.

Unfortunately, these criteria cannot be optimized simultaneously. Figure 4.2 shows three different drawings of the same representation, each optimizing a different criterion.

None of these drawings is optimal with respect to one of the other criteria. However, in the general case, the aesthetic criteria are not independent: In most cases, a drawing with short edges will not cover a large amount of the drawing area and the maximum edge length will also be reasonably small. We will also show in our experimental study (Section 4.4) that minimizing the total edge length has a positive influence on the other two optimization goals. For area and maximum edge length, the situation is different. Since minimizing the area affects mainly the boundary of a drawing Γ, edges in the middle of Γ may be drawn unnecessarily long without any influence on the measure. Also exclusively aiming at drawing the longest edge of a drawing short admits much freedom for the lengths of the other edges that may result in a waste of drawing area.



$A(\Gamma) = 12$ $L(\Gamma) = 19$ $l_{\max}(\Gamma) = 3$ (a)

$A(\Gamma) = 18$ $L(\Gamma) = 17$ $l_{\max}(\Gamma) = 3$ (b)

$A(\Gamma) = 15$ $L(\Gamma) = 18$ $l_{\max}(\Gamma) = 2$ (c)

Figure 4.2: Three drawings of the same representation. (a) Minimum area. (b) Minimum total edge length. (c) Minimum maximum edge length

The following definition formalizes the most important optimization problems that occur during the compaction phase in orthogonal graph drawing:

Definition 4.1 (Two-dimensional compaction problems in orthogonal graph drawing). Given a simple orthogonal representation H, find an orthogonal grid drawing Γ of H of minimum

(a)	width $W(\Gamma)$	(W-COMP)
(b)	height $H(\Gamma)$	(H-COMP)
(c)	half-perimeter $P(\Gamma)$	(P-COMP)
(d)	total edge length $L(\Gamma)$	(L-COMP)
(e)	maximum edge length $l_{\max}(\Gamma)$	($l_{\max}$-COMP)
(f)	area $A(\Gamma)$	(A-COMP) .

The differences between compaction problems in VLSI and in orthogonal graph drawing take effect in particular in complexity considerations. Almost all two-dimensional compaction problems in VLSI and graph drawing are *NP*-hard optimization problems including those stated in Definition 4.1. The techniques in the proofs, however, differ considerably. Most complexity proofs for the problems exploit the fact that wires may be swapped, or equivalently, the topology of the underlying graph may be changed (Dolev, Leighton, and Trickey, 1984; Storer, 1984; Bhatt and Cosmadakis, 1987; Brandenburg, 1988; Gregori, 1989). This technique cannot be employed in a proof of complexity for the problems in Definition 4.1, since the topology of the input is fixed.

Dolev and Trickey (1981) prove *NP*-hardness for A-COMP in the case that the input graph is allowed to be disconnected. Vijayan and Wigderson (1985) mention the classification for connected graphs as an open problem. Patrignani (1999) shows that the decision version of A-COMP is *NP*-complete by a conceptionally elegant reduction from the satisfiability problem SAT. The SAT problem asks for a satisfying truth assignment for a collection of clauses over a set of Boolean variables, *i.e.*, an assignment of truth values so that at least one literal in each clause is true. See (Cook, 1971) for an *NP*-completeness proof of SAT.

In order to transform one instance to the other, Patrignani introduces a powerful device, the so-called sliding-rectangles gadget. Given a formula φ with n variables and m clauses, he shows how to construct the gadget $H_A(\varphi)$ which admits an orthogonal drawing of area $K_A(\varphi) = (9n + 4)(9m + 7)$ if and only if φ is satisfiable. The proofs of *NP*-completeness for the decision versions of $l_{\max}$-COMP and L-COMP are similar reductions from SAT.

Motivated by compaction strategies in VLSI design, the following one-dimensional problems appear in the context of improvement heuristics for the problems in Definition 4.1.

Definition 4.2 (One-dimensional compaction problems in orthogonal graph drawing). Given a drawing $\Gamma = (\Gamma_x, \Gamma_y)$ of a simple orthogonal representation H, find an orthogonal grid drawing

$$\Gamma' = \begin{cases} (\Gamma'_x, \Gamma_y) & \text{(horizontal one-dimensional compaction)} \\ (\Gamma_x, \Gamma'_y) & \text{(vertical one-dimensional compaction)} \end{cases}$$

of H of minimum

(a)	width $W(\Gamma)$	(W-ICOMP, horizontal compaction)
(b)	height $H(\Gamma)$	(H-ICOMP, vertical compaction)
(c)	total edge length $L(\Gamma)$	(L-ICOMP)
(d)	maximum edge length $l_{\max}(\Gamma)$	($l_{\max}$-ICOMP)

According to the above definition, only either x- or y-coordinates can change at a time; clearly, minimizing width makes only sense during horizontal compaction and minimizing height during vertical compaction. Furthermore, since the definition affects only one dimension, minimizing half-perimeter or area is identical—from a one-dimensional point of view—to minimizing width or height, depending on the direction of the compaction. It is important to notice that the problem of minimizing one-dimensional width, height and edge length is different from the related two-dimensional problems. The restricted problems of Definition 4.2 are solvable in polynomial time; we will also show this in Section 4.2.5 where we present a generic improvement heuristics for the two-dimensional compaction problems. We will describe algorithms in this scheme that repeatedly solve one-dimensional subproblems to optimality.

4.2 Combinatorial Characterization

In our combinatorial approach to the compaction problems of the previous section, constraint graphs play the central role. In this section, we define the placement graphs as a

pair of special constraint graphs that are closely connected to the two-dimensional compaction problems in orthogonal graph drawing through the notion of *segments*. Segments correspond to maximal sets of equally-oriented consecutive edges and are represented by nodes in the placement graphs. We show that placement graphs with certain arc sets—the *shape graphs*—correspond exactly to orthogonal representations and thus define a different notion of orthogonal shape. We introduce the important concept of completeness—a combinatorial property based on paths and cycles in a pair of placement graphs; a central theorem shows that every pair of feasible assignments for a pair of complete placement graphs which are defined on an orthogonal representation H leads to an orthogonal grid drawing for H. Corollaries of this theorem yield exact polynomial-time algorithms for the compaction problems in the special case that the shape graphs are complete. We review previously known constructive and improvement heuristics for the compaction problems and study the one-dimensional compaction scheme. Our observations motivate the reformulations of the originally geometric problems as combinatorial problems: Given a pair of shape graphs, the new task consists of optimizing over the *complete extensions* of these constraint graphs.

4.2.1 Segments and Placement Graphs

As seen in Section 2.2, the shape of a simple orthogonal drawing Γ of a four-planar graph $G = (V, E)$ induces a partition of half-edges into the sets $\vec{E}_r$, $\vec{E}_l$, $\vec{E}_u$, and $\vec{E}_d$ which contain the half-edges pointing right, left, up, and down, respectively. This defines also a partition of the edge set E into a set of horizontal edges E_h and a set of vertical edges E_v.

Definition 4.3 (Segment). Let H be a simple orthogonal representation of a four-planar graph $G = (V, E)$. We call the vertex sets $S \in 2^V$ of the maximally connected components in $(V, \vec{E}_r)$ the *horizontal segments* $\mathcal{S}_h$ of G. Similarly, we define the set of *vertical segments* $\mathcal{S}_v$ as the vertex sets of the maximally connected components in $(V, \vec{E}_u)$. We refer by $\mathcal{S} = \mathcal{S}_h \cup \mathcal{S}_v$ to the set of all segments.

Since each vertex belongs to a unique maximally connected component, both the horizontal and the vertical segments are a partition of the vertex set. We denote by $\mathrm{hor}(v)$ and $\mathrm{vert}(v)$ the unique horizontal and vertical segment a vertex v belongs to. We assume as a convention that the sequence of vertices in the segments reflects their left-to-right and bottom-to-top order.

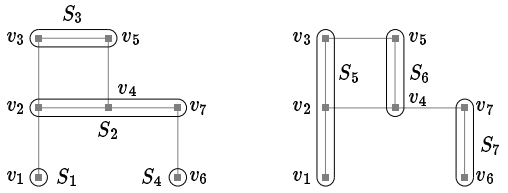
Figure 4.3 shows an orthogonal grid drawing and the corresponding segments.

Observe that assigning a y-coordinate to each horizontal segment and an x-coordinate to each vertical segment determines the coordinates of all the vertices in G. Whether the coordinates satisfy the criteria of a drawing or not, however, depends on the values we assign to $\mathcal{S}_h$ and $\mathcal{S}_v$. We will investigate this relationship in the rest of this section.

The following lemma implies that the total number of segments is $2|V| - |E|$:

Lemma 4.1. *The number of horizontal and vertical segments is*

$$|\mathcal{S}_h| = |V| - |E_h| \quad \textit{and} \quad |\mathcal{S}_v| = |V| - |E_v| \ .$$



$$
\begin{aligned}
S_1 &= \{v_1\} & \qquad S_5 &= \{v_1, v_2, v_3\}\\
S_2 &= \{v_2, v_4, v_7\} & S_6 &= \{v_4, v_5\}\\
S_3 &= \{v_3, v_5\} & S_7 &= \{v_6, v_7\}\\
S_4 &= \{v_6\}
\end{aligned}
$$

Figure 4.3: Horizontal and vertical segments of a simple orthogonal grid drawing

Proof. Induction on the number of edges $|E| = |E_h| + |E_v|$.

Induction basis ($|E| = 0$): The maximally connected components consist of single vertices. Thus, $|\mathcal{S}_h| = |\mathcal{S}_v| = |V|$.

Induction step ($|E| \to |E'| = |E| + 1$): Without loss of generality, let the additional edge be horizontal; it causes a union of two horizontal segments. Then $|V'| = |V|$, $|E'_h| = |E_h| + 1$ and $|E'_v| = |E_v|$. It follows

$$
\begin{aligned}
|\mathcal{S}'_v| &= |\mathcal{S}_v| = |V| - |E_v| = |V'| - |E'_v|\\
|\mathcal{S}'_h| &= |\mathcal{S}_h| - 1 = |V| - |E_h| - 1 = |V'| - |E'_h| \ .
\end{aligned}
$$

The total number of segments is $|V| - |E_v| + |V| - |E_h| = 2|V| - |E|$. ∎

In the rest of this chapter, segments and relations between segments play a key role. Therefore, we introduce the pair of *placement graphs* as a generic structure that is able to express these relationships. In the following we will study several kinds of placement graphs that are closely related to the compaction problems as described in Section 4.1.

Definition 4.4 (Placement graphs, coordinate assignment). A pair of placement graphs consists of two constraint graphs D_x and D_y and corresponds to a simple orthogonal representation H. The node set of D_x is the set of vertical segments $\mathcal{S}_v$, the node set of D_y consists of the horizontal segments $\mathcal{S}_h$. Let c_x and c_y be feasible assignments for D_x and D_y, respectively. We call the pair (c_x, c_y) a *coordinate assignment* for the pair of placement graphs (D_x, D_y).

4.2.2 Shape Graphs

In this section we will define the *shape graphs* that provide a different notion of orthogonal shape. Shape graphs are a special pair of placement graphs and correspond to a given simple orthogonal representation. We will show that there is a unique pair of shape graphs for each representation H and that this pair can be constructed in linear time.

Definition 4.5 (Shape graphs). Let $\vec{E}_r$, $\vec{E}_l$, $\vec{E}_u$ and $\vec{E}_d$ be the partition of half-edges induced by a simple orthogonal representation H. A pair of shape graphs $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ is a pair of placement graphs, *i.e.*, $N_x = \mathcal{S}_v$, $N_y = \mathcal{S}_h$. Its arc sets are

$$A_x = \{(\text{vert}(v), \text{vert}(w)) \mid (v, w) \in \vec{E}_r\}$$
$$A_y = \{(\text{hor}(v), \text{hor}(w)) \mid (v, w) \in \vec{E}_u\} \ .$$

All arc weights are equal to one, *i.e.*,

$$w_h(a) = 1 \quad \text{for all } a \in A_h$$
$$w_v(a) = 1 \quad \text{for all } a \in A_v \ .$$

Remark 4.1. The unit arc weights express the minimum edge length if the aim is to compute an orthogonal grid drawing. In a more general definition, the weights correspond to arbitrary minimum distance requirements. Unless otherwise stated, we will assume unit weights as in the above definition throughout this chapter.

Figure 4.4 shows a pair of shape graphs, continuing the example of Figure 4.3. Note that shape graphs are multigraphs.

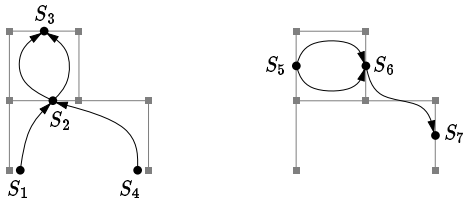


Figure 4.4: Shape graphs for the example in Figure 4.3

The following lemma shows that shape graphs are an equivalent way of describing the shape of an orthogonal representation and can be built in linear time.

Lemma 4.2. *Given a simple orthogonal representation H with n vertices, the corresponding shape graphs are unique and can be computed in time $O(n)$.*

Proof. Constructing the pair of shape graphs of H is straightforward. Using the partition of half-edges, we can build the graphs as described by Definitions 4.3, 4.4 and 4.5: First, we determine the segments by computing maximally connected components which takes linear time. Then, for each half-edge $\vec{e}$ in $\vec{E}_r \cup \vec{E}_u$, we link the corresponding nodes according to the direction of the original edges. This takes also time $O(n)$ since H represents a planar graph. ∎

In the following we will refer by $\sigma(H)$ to the unique pair of shape graphs D_x and D_y that corresponds to the orthogonal representation H. The following lemma shows an important property of shape graphs and implies that shape graphs are acyclic:

Lemma 4.3. *Shape graphs are upward planar.*

Proof. The upward planar embedding is defined by construction: Clearly, all arcs in D_x and D_y point in the same direction since they correspond to half-edges of either rightward or upward direction. We can imagine the arcs running along the half-edges which defines the upward planar embedding (see Figure 4.4). ∎

4.2.3 Completeness

Consider a pair of placement graphs (D_x, D_y). We establish a relationship between the nodes in the two constraint graphs by observing that each horizontal segment is bounded by two (not necessarily different) vertical segments and vice versa. We express this fact by the following definition.

Definition 4.6 (Limits). Let $S = \{v_1, \ldots, v_k\}$ be a segment and assume the indices reflect the left-to-right and bottom-to-top order induced by the shape. We define four vectors $l \in \mathcal{S}_v^{\mathcal{S}}$, $r \in \mathcal{S}_v^{\mathcal{S}}$, $b \in \mathcal{S}_h^{\mathcal{S}}$, and $t \in \mathcal{S}_h^{\mathcal{S}}$, the *left*, *right*, *bottom*, and *top limit*, as follows:

$$\begin{aligned} l_S &= \text{vert}(v_1) & r_S &= \text{vert}(v_k) \\ b_S &= \text{hor}(v_1) & t_S &= \text{hor}(v_k) \quad . \end{aligned}$$

Table 4.1 shows the limits of the segments in the example of Figures 4.3 and 4.4. A general property of the limits can be observed: A vertical segment is bounded from the left and the right by itself, as well as a horizontal segment is bounded from below and from top by itself, *i.e.*,

$$\begin{aligned} l_S = r_S = S \quad &\text{for all } S \in \mathcal{S}_v \\ b_S = t_S = S \quad &\text{for all } S \in \mathcal{S}_h \quad . \end{aligned}$$

i	l_{S_i}	r_{S_i}	b_{S_i}	t_{S_i}
1	S_5	S_5	S_1	S_1
2	S_5	S_7	S_2	S_2
3	S_5	S_6	S_3	S_3
4	S_7	S_7	S_4	S_4
5	S_5	S_5	S_1	S_3
6	S_6	S_6	S_2	S_3
7	S_7	S_7	S_4	S_2

Table 4.1: Limits of the segments of Figure 4.3

Definition 4.7 (Completeness). A pair of placement graphs is *complete* if and only if

(a) Both graphs are acyclic;

(b) For all pairs of segments (I, J) with $I \cap J \neq \varnothing$ the constraint graphs contain one of the four following paths:

$$r_I \xrightarrow{*} l_J \qquad r_J \xrightarrow{*} l_I$$
$$t_I \xrightarrow{*} b_J \qquad t_J \xrightarrow{*} b_I \ .$$

If one of the upper two paths exists, we call (I, J) *horizontally separated*, if one of the lower two paths exists, we call the pair *vertically separated*. The pair is *separated* if it is horizontally or vertically separated.

Remark 4.2. The definition of completeness relies on the assumption that the arc weights are non-negative. A more general definition that also holds for general arc weights requires the absence of positive cycles instead of acyclicity in Definition 4.7(a) and demands one of the four paths in Definition 4.7(b) to be of positive weight. We will need this extended notion of completeness for the *labeling graphs* and for the *shape and labeling graphs* in Chapters 5 and 6, see also Remark 5.2 and Definition 6.4 on pages 129 and 162.

Lemma 4.4. *A pair of placement graphs is complete if and only if it is acyclic and all segment pairs of opposite direction are separated.*

Proof. The forward direction of the proof is trivial. For the backward direction, assume that there is a pair $(I, J) \in \mathcal{S} \times \mathcal{S}$ which is not separated and every pair of opposite directions is separated. Consider the following cases:

Case 1: $(I, J) \in \mathcal{S}_h \times \mathcal{S}_v$ or $(I, J) \in \mathcal{S}_v \times \mathcal{S}_h$. This is a contradiction to the assumption.

Case 2: $(I, J) \in \mathcal{S}_h \times \mathcal{S}_h$. Let $L = \{l_I, r_I, l_J, r_J\}$ be the horizontal limits of I and J. For each vertical segment $\ell \in L$, we define $h(\ell)$ as the segment which is limited by ℓ and $p(\ell)$ as the other horizontal segment. More precisely:

ℓ	$h(\ell)$	$p(\ell)$
l_I	I	J
r_I	I	J
l_J	J	I
r_J	J	I

Now consider the four segment pairs of opposite direction in $\bigcup_{\ell \in L} \{(\ell, p(\ell))\}$. Note that $b_{h(\ell)} = h(\ell) = t_{h(\ell)}$ and $b_l \xrightarrow{*} h(\ell) \xrightarrow{*} t_\ell$ for all $\ell \in L$. It follows that none of the pairs is vertically separated, since

$$\begin{aligned}
& t_{p(\ell)} \xrightarrow{*} b_\ell && \vee && t_\ell \xrightarrow{*} b_{p(\ell)} \\
\Leftrightarrow \quad & p(\ell) \xrightarrow{*} b_\ell && \vee && t_\ell \xrightarrow{*} p(\ell) \\
\Rightarrow \quad & p(\ell) \xrightarrow{*} h(\ell) && \vee && h(\ell) \xrightarrow{*} p(\ell) \\
\Leftrightarrow \quad & I \xrightarrow{*} J && \vee && J \xrightarrow{*} I,
\end{aligned}$$

which is a contradiction to the separation of I and J.

For the horizontal part, note that $l_\ell = r_\ell = \ell$ for all $\ell \in L$. The horizontal separation of the segment pairs $(\ell, p(\ell))$ for $\ell \in L$ is expressed by

$$\begin{aligned} & r_\ell \xrightarrow{*} l_{p(\ell)} \quad \vee \quad r_{p(\ell)} \xrightarrow{*} l_\ell \\ \Leftrightarrow \quad & \ell \xrightarrow{*} l_{p(\ell)} \quad \vee \quad r_{p(\ell)} \xrightarrow{*} \ell. \end{aligned}$$

For each ℓ, one of the two terms implies the separation of I and J and can be eliminated. The remaining four terms induce two cycles in the graph, which is again a contradiction to the assumption.

Case 3: $(I, J) \in \mathcal{S}_v \times \mathcal{S}_v$. This case is similar to the previous one. ∎

Complete placement graphs have useful properties. The following central theorem shows that feasible assignments for complete placement graphs result in grid drawings for the corresponding orthogonal representation. We will show that the property of completeness characterizes exactly those placement graphs for which all feasible coordinate assignments result in a valid placement of vertices and bends of the underlying representation. We will exploit this characterization in particular to compute minimal and Δ-minimal assignments for complete shape graphs. Two corollaries of the following theorem state that this leads to exact polynomial-time algorithms for most of the two-dimensional compaction problems of Definition 4.1 in the case of complete shape graphs. Although the general case, in which the shape graphs are not complete, is far more frequent, these results build the fundament of new exact algorithms in the next section where we will exploit Theorem 4.1 and its corollaries algorithmically.

Theorem 4.1. *Given a simple orthogonal representation H of a four-planar graph $G = (V, E)$ and a pair of complete placement graphs $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ for H. Every coordinate assignment (c_x, c_y) for D_x and D_y results in an orthogonal grid drawing for H.*

Proof. Since both D_x and D_y are acyclic, Theorem 3.1 guarantees the existence of feasible assignments; let (c_x, c_y) be such a coordinate assignment. We construct an orthogonal grid drawing Γ as follows. For all $v \in V$ we set

$$\Gamma(v) = \big(c_x(\mathrm{vert}(v)), c_y(\mathrm{hor}(v)\big)$$

and draw each edge $e = (v, w) \in E$ as a line segment from $\Gamma(v)$ to $\Gamma(w)$.[1]

We show that Γ is indeed an orthogonal grid embedding by verifying the following three criteria, see also Definition 2.1:

1. For $v \neq w \in V$ it holds $\Gamma(v) \neq \Gamma(w)$.
   Assume otherwise, *i.e.*, $v \neq w$ but both vertices occupy the same grid point. Let $I = \mathrm{hor}(v)$, $J = \mathrm{vert}(v)$, $K = \mathrm{hor}(w)$ and $L = \mathrm{vert}(w)$ as in Figure 4.5. Note

[1] Note that H is simple.

that the drawing does not reflect the actual coordinates $c_x(J) = c_x(L)$ and $c_y(I) = c_y(K)$. Since the pair of placement graphs is complete, all pairs of segments must be separated, in particular the segments J and L. We have the following cases (see Definition 4.7):

Case 1: $r_J = J \xrightarrow{*} L = l_L$.
Because of Observation 3.2 and feasibility of c_x for D_x we must have

$$c_x(L) - c_x(J) > 0 \ ,$$

a contradiction to the horizontal alignment of v and w.

Case 2: $r_L = L \xrightarrow{*} J = l_J$. This case is symmetric to Case 1 and also leads to a contradiction.

Case 3: $t_L \xrightarrow{*} b_J$.
The path requires that $c_y(b_J) > c_y(t_L)$. The definition of limits implies $c_y(b_J) < c_y(J)$ and $c_y(L) < c_y(t_L)$. Note that the limits reflect the left-to-right and bottom-to-top order. Combining the inequalities results in the contradiction

$$c_y(b_J) < c_y(J) = c_y(L) < c_y(t_L) < c_y(b_J) \ .$$

Case 4: $t_J \xrightarrow{*} b_L$. This case is symmetric to Case 3 and also leads to a contradiction.

Since all four possible cases lead to contradictions, a coordinate assignment for complete placement graphs maps different vertices to different grid points.

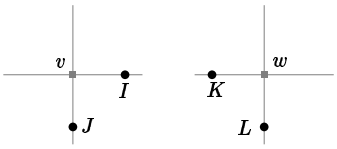
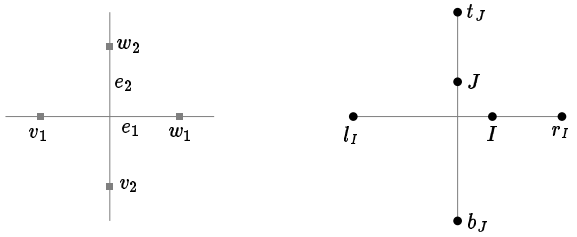


Figure 4.5: Placement of vertices in the proof of Theorem 4.1

2. Γ maps each edge $(v, w) \in E$ to a path on the grid with endpoints $\Gamma(v)$ and $\Gamma(w)$. By construction, Γ maps each edge $e = (v, w)$ to a line segment from $\Gamma(v)$ to $\Gamma(w)$. For a horizontal edge $(v, w) \in E_h$ we have $\text{hor}(v) = \text{hor}(w)$, hence v and w have the same y-coordinate. Similar reasoning shows that endpoints of vertical edges have the same x-coordinates.

3. The line segments for edges overlap only at common endpoints.
Assume otherwise, *i.e.*, there are two edges $e_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2)$ that cross in Γ. Lemma 4.4 allows us to assume without loss of generality that e_1 is horizontal and e_2 vertical as depicted in Figure 4.6.



(a) Situation in Γ (b) Situation in placement graphs

Figure 4.6: Crossing edges

Similar to the part of the proof for the vertex placement, we consider a segment pair (I, J). Since e_1 and e_2 cross, we must have

$$c_x(l_I) < c_x(J) < c_x(r_I) \tag{4.1}$$
$$c_y(b_J) < c_y(I) < c_y(t_J) \tag{4.2}$$

Separation of I and J requires at least one of the four cases:

Case 1: $r_I \xrightarrow{*} l_J = J$.
Feasibility of c_x and Observation 3.2 yield $c_x(J) > c_x(r_I)$, a contradiction to (4.1).

Case 2: $r_J = J \xrightarrow{*} l_I$.
We get $c_x(l_I) > c_x(J)$, a contradiction to (4.1).

Case 3: $t_I = I \xrightarrow{*} b_J$.
We get $c_y(b_J) > c_y(I)$, a contradiction to (4.2).

Case 4: $t_J \xrightarrow{*} I = b_I$.
We get $c_y(I) > c_y(t_J)$, a contradiction to (4.2).

Since all cases lead to contradictions, no edges share the same paths on the grid unless they have common endpoints. ∎

An interesting class of orthogonal representations are those having complete shape graphs. Two corollaries of Theorem 4.1 demonstrate how to solve the compaction problems to optimality for this class of representations. A third corollary justifies to concentrate on local completeness in order to obtain feasible solutions of the compaction problems.

Definition 4.8 (Complete orthogonal representation). An orthogonal representation H is *complete* if the corresponding shape graphs $\sigma(H)$ are complete.

Corollary 4.1. *Let H be an orthogonal representation of a four-planar graph $G = (V, E)$. If H is complete, an orthogonal grid drawing Γ of minimum width $W(\Gamma)$, height $H(\Gamma)$*

and area $A(\Gamma)$ for H can be computed in time $O(|V|)$. The following bounds hold:

(a) $W(\Gamma) \leq |V| - |E_v|$
(b) $H(\Gamma) \leq |V| - |E_h|$
(c) $P(\Gamma) \leq 2|V| - |E|$
(d) $A(\Gamma) = O(|V|^2)$

Proof. Let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be the corresponding pair of shape graphs $\sigma(H)$ and let E_h and E_v be the horizontal and vertical edges induced by H. Theorem 4.1 suggests an algorithm for computing optimal orthogonal grid drawings for representations with complete shape graphs: For each of the graphs D_x and D_y we use Algorithm 3.2 to determine a minimal assignment c_x and c_y. Since both D_x and D_y are planar (Lemma 4.3), we have $A_x = O(N_x)$ and $A_y = O(N_y)$. Furthermore, we know that $|N_x| = |V| - |E_v| = O(|V|)$ and $|N_y| = |V| - |E_h| = O(|V|)$ by Lemma 4.1. It follows $A_x = O(|V|)$ and $A_y = O(|V|)$, hence Algorithm 3.2 takes linear time by Theorem 3.3.

Since the assignments are minimal and all arcs have unit weight, the span in c_x and c_y is bounded, more precisely:

$$\begin{aligned} \operatorname{sp}(c_x) &\leq |N_x| = |V| - |E_v| \\ \operatorname{sp}(c_y) &\leq |N_y| = |V| - |E_h| \quad . \end{aligned}$$

In the corresponding grid drawing Γ, these spans are equal to the width $W(\Gamma)$ and the height $H(\Gamma)$. This proves statements (a), (b), and (c). The minimality of the spans implies the minimality of width, height and half-perimeter. Since both width and height are minimal, also the area $A(\Gamma)$ must be minimal; the bound for the area follows by (a) and (b). ∎

Corollary 4.2. *Given an orthogonal representation H of a four-planar graph $G = (V, E)$. If H is complete, an orthogonal grid drawing Γ of minimum total edge length $L(\Gamma)$ can be computed in time $O(|V|^2(\log|V|)^2)$. The bounds for $W(\Gamma)$, $H(\Gamma)$, $P(\Gamma)$ and $A(\Gamma)$ are the same as in Corollary 4.1.*

Proof. The proof is similar to the proof of Corollary 4.1. Instead of minimal assignments, we use Algorithm 3.3 to compute Δ-minimal assignments using a minimum-cost flow algorithm. Each arc in D_x corresponds to a half-edge in $\vec{E}_r$ and each arc in D_y corresponds to a half-edge in $\vec{E}_u$; the distances of the arcs in the assignment for the shape graphs correspond to the lengths of the respective edges. Thus, the two Δ-minimal assignments result in a drawing Γ of minimum total edge length $L(\Gamma)$. Since shape graphs are planar (Lemma 4.3), the number of arcs in D_x and D_y is linear in the size of H and Algorithm 3.3 has running time $O\big((|V|\log|V|)(|V| + |V|\log|V|)\big) = O(|V|^2(\log|V|)^2)$. ∎

Placement graphs describe segments and their relative positioning relations for a given orthogonal shape with the goal to characterize properties of planar orthogonal drawings with that shape. We exploit the structure of these drawings and show that local completeness suffices to construct drawings which simplifies some of the proofs in the next sections.

Corollary 4.3. *Let (D_x, D_y) be a pair of placement graphs for a simple orthogonal representation H. For a face f in H, let $\mathcal{S}(f)$ denote the set of segments containing the edges on the boundary of f. If the pair of placement graphs does not contain cycles and if for every face f the segments pairs $(I, J) \in \mathcal{S}(f) \times \mathcal{S}(f)$ are separated, then every coordinate assignment for (D_x, D_y) results in an orthogonal grid drawing for H.*

Proof. The absence of cycles guarantees a feasible coordinate assignment (c_x, c_y) as in the proof of Theorem 4.1. Observation 3.1 states that (c_x, c_y) is feasible for the constraint subgraphs that correspond to the single faces of H. Since the subgraphs are complete, Theorem 4.1 ensures a feasible subdrawing for every face. Consider the dual graph of G in which every face is represented by a vertex and linked by an edge to the vertices corresponding to neighboring faces. Let the *distance* of two faces f and g be the length of the shortest path between the corresponding vertices in the dual graph, and let f be an arbitrary face in H. Clearly, all distances are finite. We continue the proof by induction on the neighboring faces of f and consider the subdrawings for faces of distance at most k from f. In case $k = 0$ the set of faces consists only of f and the local completeness ensures that the coordinate assignment results in a drawing for f. Assume that the assignment yields drawings for distances up to k and consider the faces of distance $k + 1$. The subdrawings of these faces share common edges with the other faces. These edges must be drawn identical due to the common assignment (c_x, c_y). Informally speaking, (c_x, c_y) pastes the subdrawings together at their points of contact. The result is a drawing for faces of distance up to $k + 1$. ∎

4.2.4 Constructive Heuristics

In this section we present *constructive heuristics* for the two-dimensional compaction problems. Algorithms in this category aim at producing a first feasible solution for a given orthogonal shape.

In the area of VLSI design, Vijayan and Wigderson (1985) consider the problem of finding an orthogonal drawing for a rectilinear graph (V, A, λ) which describes the shape of a four-graph $G = (V, E)$ by means of a vertex ordering relation, see Definition 2.4. This concept is equivalent to the notion of orthogonal representation as seen in Section 2.2. The authors present an $O(|V|^2)$ algorithm that constructs orthogonal grid drawings for rectilinear graphs. Their algorithm constitutes a first constructive heuristics for the compaction problems of Section 4.1.

Tamassia (1987) and Hoffmann and Kriegel (1988) independently improve this result: both papers describe a linear time strategy which is based on dissecting the faces of the graph. The common idea is to transform the given orthogonal representation H into an auxiliary representation H' for which it is easier to find solutions to the compaction problems. The transformation step consists of adding artificial vertices and edges to H without changing the shape of the original representation. This property ensures that a drawing Γ' for H' contains a drawing Γ for H: The drawing for the original graph results from deleting the artificial vertices and edges in Γ'.

While the above methods dissect the internal faces into subfaces of rectangular shape, Bridgeman *et al.* (2000) describe two similar constructive heuristics based on the concept

of turn-regularity. Informally, the shape of a face f is turn-regular if there are no opposite corners in f. The heuristics dissect non-turn-regular faces in rectangular or turn-regular subfaces.

In the following, we summarize the two general types of constructive heuristics for the compaction problems in orthogonal graph drawing: First, we describe the traditional method of Tamassia (1987) and some of its variants. These strategies construct the auxiliary representation H' by dissecting each internal face of H into a set of rectangle-shaped subfaces. We show that the shape graphs that correspond to H' are complete and therefore the corresponding compaction problems are solvable to optimality in polynomial time. We continue by describing the turn-regularity-based approach that also leads to a polynomial-time compaction algorithm in case all faces of H are turn-regular.

We want to emphasize that the papers (Vijayan and Wigderson, 1985), (Tamassia, 1987), (Hoffmann and Kriegel, 1988) and (Bridgeman *et al.*, 2000) already contain results leading to polynomial time compaction algorithms for the respective classes of orthogonal representations. Yet, the techniques and proofs presented in this section are based on the combinatorial properties of constraint graphs and contribute to the construction of the hierarchy of orthogonal representations as it is introduced at the end of Section 4.2.6. We will develop exact algorithms for the different classes of orthogonal representations in the hierarchy. Our new algorithms solve the compaction problems to provable optimality in polynomial time for instances in all but the last layer of the hierarchy.

Rectangle-Based Dissection

Due to their simplicity, rectangle-based dissection methods are the most common constructive heuristics for orthogonal compaction in graph drawing. They produce the auxiliary representation H' by dissecting each internal face of the given simple orthogonal representation H into a set of faces each of which has rectangular shape and by transforming the shape of the external face so that it corresponds to the complement of a rectangle.[2]

Let $S(f)$ be the circular bit string that describes the orthogonal shape of face f, see the definition on page 20. Algorithm 4.1 repeatedly cuts off rectangle-shaped subfaces from each face and updates the bit string accordingly until none of the faces contains the pattern 100. We refer to this operation as Operation 1, see Figure 4.7(a). This process transforms the shape of internal faces into rectangles. The shape of the external face f_0, however, may still not correspond to the complement of a rectangle, but has the form $S(f_0) = 1(01)^*1(01)^*1(01)^*1(01)^*$. Note that bit strings may be shifted circularly. The shape characterized by $S(f_0)$ corresponds to the complementary shape of a *convex face* as defined in Hoffmann and Kriegel (1988). Similar to the above process, a second operation replaces the substring 101 by 1 and yields the complementary shape, see Figure 4.7(b). Both sequences of operations can be realized in time $O(|f|)$ for each face f by stacking the one-bits during the search process; this observation shows that Algorithm 4.1 needs $O(n)$ time to dissect a simple representation with n vertices. A proof of correctness for this rectangular dissection method can be found in (Di Battista *et al.*, 1999b); Figure 4.8

[2] Our description of the rectangular dissection methods in this section is based on the article (Tamassia, 1987). In addition, we mention some improvements and variants.

illustrates the result of Algorithm 4.1 for an example.[3]

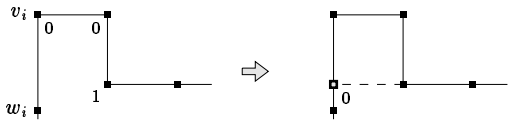
Algorithm 4.1 *Rectangular dissection*

Input: Simple orthogonal representation H with face set F

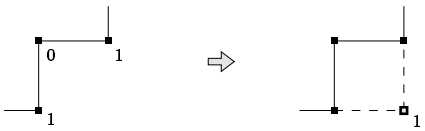
Output: Simple orthogonal representation H' with

$$S(f) = \begin{cases} 0000 & f \text{ internal} \\ 1111 & f \text{ external} \end{cases}$$

1: **for all** $f \in F$ **do**
2: **while** $S(f)$ contains 100 **do**
3: search for pattern 100 in $S(f)$;
4: perform Operation 1 and update shape (see Figure 4.7(a));
5: replace 100 by 0;
6: **while** $S(f_0)$ contains 101 **do**
7: search for pattern 101 in $S(f_0)$;
8: perform Operation 2 and update shape (see Figure 4.7(b));
9: replace 101 by 1;



(a) Operation 1



(b) Operation 2

Figure 4.7: Two dissection operations

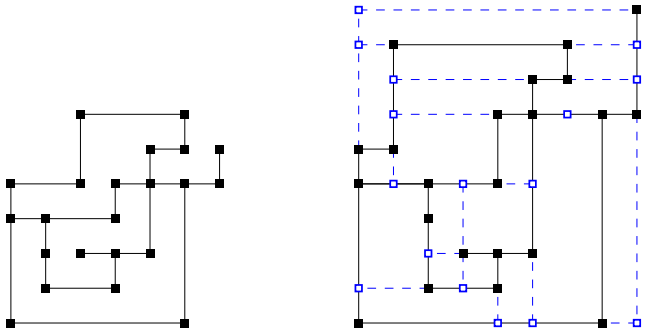
Several variants of Algorithm 4.1 exist:

- As Lemma 4.5 will show, the special treatment for the external face f_0 is not necessary. It is safe to perform only Operation 1 on f_0.
- Operation 1 may be extended by looking at more complex patterns. In addition to 100, one may also look for 1001 and replace it by ε or for 10001 and replace it by 0 and so on.

[3] The drawings may be misleading: The method works at the level of the representation; it should not be overseen that the coordinates have not yet been assigned.

– Similar to (4.2.4), the number of artificial vertices can be reduced by connecting artificial edges to ordinary vertices if this is possible. If, *e.g.*, in Figure 4.7(a) the edge $e_i = (v_i, w_i)$ after the pattern forms an angle $a_i \in \{180, 270, 360\}$, the vertex w_i may play the role of an artificial vertex.

– Mühlenbacher (2001) describes a still linear-time variant in which the direction of all artificial edges is either exclusively horizontal or exclusively vertical.



(a) (b)

Figure 4.8: The rectangle-based dissection method (standard variant). (a) Original representation H, (b) Transformed representation H'. Dashed lines and empty squares represent artificial edges and vertices

The following definition characterizes orthogonal representations in which each internal face has rectangular shape. The rectangle-based dissection methods transform the original representation H into an auxiliary representation H' which satisfies this criterion and includes the shape of H. Removing the artificial vertices and edges while adapting the shape accordingly results in the original representation.

Definition 4.9 (Rectangular orthogonal representation). An orthogonal representation is *rectangular* if for every face f in H the circular bit string $S(f)$ does not contain the pattern 100.

An equivalent characterization is that the shape of each internal face f must be rectangular, *i.e.*, $S(f) = 0000$, and the shape of f_0 must correspond to the circular bit string $1(01)^*(01)^*(01)^*(01)^*$. The following lemma shows that rectangular representations are easy instances for the compaction problems:

Lemma 4.5. *Rectangular representations are complete.*

Proof. A *staircase* P_S is a directed path of half-edges in $\vec{E}_n \cup \vec{E}_e$ or $\vec{E}_n \cup \vec{E}_w$. Drawings of staircases must be monotone in x- and y-direction. Let H be an orthogonal representation with corresponding shape graphs $\sigma(H)$. Note that segments that are linked by a staircase

in H are separated in $\sigma(H)$. Consider Figure 4.9 which shows a rectangular representation and observe that, due to the special shape, every pair of segments I and J is linked by a staircase. ∎

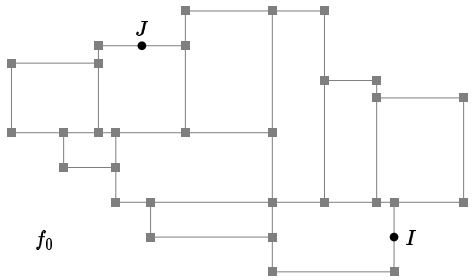


Figure 4.9: Drawing of a rectangular orthogonal representation

The completeness of rectangular representations and the fact we can use two separate assignments for complete pairs of placement graphs in order to compute a coordinate assignment yields a polynomial-time exact algorithm for the compaction problems in the case of rectangular representations, see Theorem 4.1 and its Corollaries 4.1 and 4.2. Together with a rectangle-based dissection strategy this leads to two constructive heuristics for general representations. Algorithm 4.2 summarizes the rectangle-based heuristics for the compaction problems in graph drawing.

Algorithm 4.2 *Rectangle-based constructive heuristics*

Input: Simple orthogonal representation H

Output: Orthogonal grid drawing for H

1: construct auxiliary representation H' with a rectangle-based dissection method;
2: compute minimal or Δ-minimal coordinate assignment for $\sigma(H')$;
3: compute coordinates for H';
4: remove artificial vertices and edges;

Theorem 4.2. *Algorithm 4.2 constructs an orthogonal grid drawing Γ for a simple representation H with n vertices*

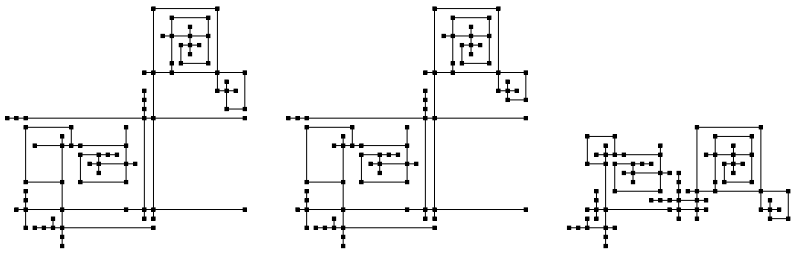
$$\text{in time} \quad \begin{cases} O(n) & \text{when using minimal assignments in line 2.} \\ O(n^2(\log n)^2) & \text{when using } \Delta\text{-minimal assignments in line 2 .} \end{cases}$$

In both cases, the bounds $W(\Gamma) = O(n)$, $H(\Gamma) = O(n)$, $P(\Gamma) = O(n)$ and $A(\Gamma) = O(n^2)$ hold.

Proof. The rectangular dissection methods run in linear time and result in an auxiliary representation H' by introducing only a linear number of artificial vertices (Tamassia,

1987; Hoffmann and Kriegel, 1988). Let $n' = O(n)$ be the number of vertices in H'. According to Lemma 4.5, the shape graphs $\sigma(H')$ are complete, and by Corollaries 4.1 and 4.2 we can construct optimal orthogonal grid drawings for H' in time $O(n')$ or $O\big((n')^2(\log n')^2\big)$, respectively. Removing the artificial objects takes linear time, the above running times for Algorithm 4.2 follow. The bounds follow from $n' = O(n)$ and the bounds in Corollaries 4.1 and 4.2. ∎

Obviously, the artificial vertices and edges impose additional constraints on the geometry which leads, in general, to suboptimal total edge length and area in the resulting drawing. Figure 4.10 shows this behavior by means of a larger example and demonstrates that rectangle-based dissection heuristics alone are not suitable for practical purposes.



(a) (b) (c)

Figure 4.10: The two rectangle-based dissection methods, illustrated with a larger example. Assignment by (a) topological order assignment and (b) minimum-cost flow assignment, compared to optimal drawing w.r.t. total edge length (c)

Remark 4.3. As Figure 4.10 shows, the drawings produced by computing Δ-minimal assignments in Algorithm 4.2 need not be better than the corresponding drawings resulting from minimal assignments. They can even be worse—this is possible if artificial edges are short at the cost of real edges. In general, using Δ-minimal assignments does not pay off for rectangular orthogonal representations: The special structure of the corresponding shape graphs does not leave much freedom for the values in a coordinate assignment.

Turn-Regularity-Based Dissection

Bridgeman *et al.* (2000) present another, more sophisticated, approach to produce an auxiliary representation H' for which polynomial-time compaction algorithms exist. Using the concept of *turn-regularity*, the authors manage to introduce a significantly lower number of artificial vertices and edges[4] which leads to better drawings than the rectangle-based

[4] See also the experimental comparison on the numbers of inserted artificial edges in Table 4.5 on page 110.

techniques. Let f be a face in H. With every occurrence of a vertex v on the boundary of f, zero, one, or two *corners* are associated with v, depending on the angle internal to f between the edges preceding and following v. An angle of 180° corresponds to no corner, angles of 90° and 270° correspond to one corner, and a 360° angle corresponds to two corners. For every ordered pair of corners (c_i, c_j) associated with vertices of f, let $\rho(c_i, c_j)$ be the difference of the left and right turns along the boundary of f between c_i (included) and c_j (excluded). The value $\rho(c_i, c_j)$ defines the net angle between the edges preceding the vertices associated with c_i and c_j. Two corners at angles of at least 270 degrees are *kitty corners* if $\rho(c_i, c_j) = 2$ or $\rho(c_j, c_i) = 2$. Figure 4.11 shows the 18 different kinds of kitty corners.

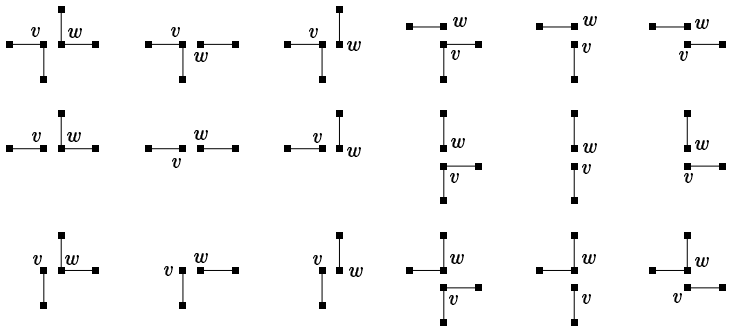


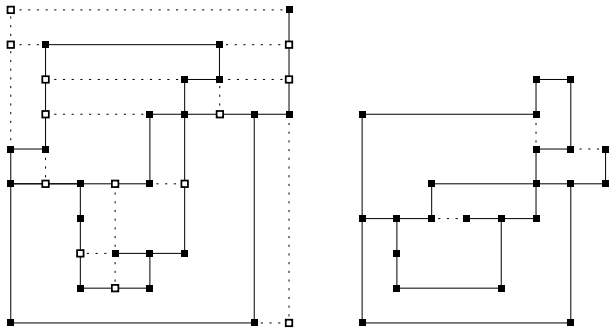
Figure 4.11: Possible configurations of kitty corners

A face of an orthogonal representation is *turn-regular* if it has no kitty corners. Clearly, rectangular faces have this property. A representation is called *turn-regular*, if all its faces are turn-regular. The turn-regularity of a representation H can be tested by relating it to the theory of upward planar graphs: Let H_r be the directed graph that results from removing the half-edges $\vec{E}_s$ and $\vec{E}_l$ from H and let H_l be the graph resulting from the removal of $\vec{E}_s$ and $\vec{E}_r$. Let the embedding of H_l and H_r be determined by the embedding of the underlying four-planar graph. By definition, these oriented versions of H are upward planar. A central theorem in (Bridgeman *et al.*, 2000) states that H is turn-regular if and only if both H_r and H_l admit a unique complete saturator, *i.e.*, a set of directed edges that can be added to the directed graphs without destroying their upward planarity so that the resulting graph is a planar s-t digraph. This property can be tested in linear time.

Similar to the rectangular case, optimal orthogonal grid drawings for turn-regular representations can be found in linear time (width-, height- and area-minimization) and in time needed for a minimum-cost flow computation (total edge length minimization). This fact is the fundament of the turn-regularity-based constructive heuristics for the two-dimensional compaction problems. The overall strategy is similar to Algorithm 4.2. It is, however, not as easy to compute feasible assignments for turn-regular representations as in the rectangular case, see (Bridgeman *et al.*, 2000) for details.

The authors present two methods to transform a general representation H into a turn-regular auxiliary representation H': The first heuristics uses the rectangular dissection

method described above, but only for non-turn-regular faces in H, see Figure 4.12(a). The second heuristics recursively adds an artificial edge between each pair of kitty corners until the face has been decomposed into smaller turn-regular, but not necessarily rectangular, faces. The direction of the inserted edge (vertical or horizontal) is chosen randomly. See Figure 4.12(b).



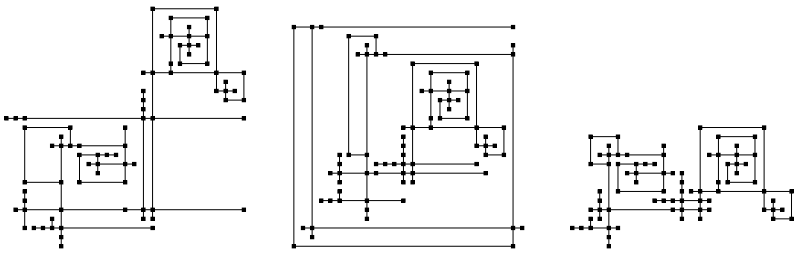
(a) (b)

Figure 4.12: The two turn-regularity-based dissection methods (same example as in Figure 4.8). Dissection in (a) rectangular and (b) turn-regular subfaces. Dashed lines and empty squares represent artificial edges and vertices.

Theorem 4.3 (Bridgeman *et al.* (2000)). *The two turn-regularity-based heuristics construct an orthogonal grid drawing Γ for a simple orthogonal representation H with n vertices in time $O(n)$ or time $O(n^{7/4}\sqrt{\log n})$, depending on the type of assignment for the auxiliary representation H'.*

Figure 4.13 shows the drawings that result from the turn-regularity-based heuristics for the example in Figure 4.10. It illustrates in particular that a lower number of artificial edges does not necessarily lead to better drawings: Artificial edge express local decisions and even a few unfortunate local decisions can destroy the proximity to an optimal drawing.

Remark 4.4. The motivation behind using turn-regular instead of rectangular auxiliary representations is to add a smaller number of artificial edges and thus to make fewer local decisions. Another strategy which operates within the orthogonalization phase has a similar goal: It aims at reducing the number of segments of the resulting orthogonal representations while preserving bend-minimality. The algorithm described in (Klau, 2001) results in a bend-minimal orthogonal representation with a minimum number of segments and is a variant of the algorithm by Tamassia (1987). The idea is to adapt the network in order to maximize the number of 180° angles that occur at vertices as a secondary optimization goal. Previous work that addresses this problem (partly among others) consists of heuristic post-processing techniques in order to straighten the orthogonal drawings (Six, Kakoulis, and Tollis, 2000; Fößmeier, Heß, and Kaufmann, 1998; Fanto, 1997). Our experiments show that the dissection strategies have to insert a significantly lower number of



(a) (b) (c)

Figure 4.13: The two turn-regularity-based heuristics applied to the example of Figure 4.10. Non-turn-regular faces are dissected in rectangular subfaces (a) or turn-regular subfaces (b), (c) shows the optimal drawing w.r.t. total edge length

artificial vertices and edges with the new variant of Tamassia's algorithm, see also Table 4.5 on page 110.

4.2.5 Improvement Heuristics

Typically, the initial drawings produced by the constructive heuristics admit considerable improvement. In this section we describe heuristics for the two-dimensional compaction problems in orthogonal graph drawing that start with a feasible solution and aim at improving the original drawing with respect to one of the aesthetic principles mentioned in Section 4.1. We refer to these heuristics as *improvement heuristics*.

This section concentrates on the most important criteria in the last phase of the topology-shape-metrics scheme: edge length and area. We will present a generic algorithmic scheme; heuristics that fit in this scheme originate in the area of VLSI design where they are known as one-dimensional compaction methods. As discussed in Section 4.1, the main focus of research concerning compaction in VLSI design is on fast methods which allow for interactive usage in a symbolic layout tool: Mostly, these tools include one-dimensional compaction algorithms.

Since the two-dimensional problems are computationally intractable for very large instances as they occur in chip layout, the idea is to treat each compaction direction separately, one at a time. During a horizontal compaction step, only x-coordinates may be changed, whereas they have to remain fixed during vertical compaction. Improvement in one dimension may allow further improvement in the other direction; an improvement heuristics solves a sequence of one-dimensional compaction problems until both compaction directions are blocked. At each step, however, the decisions are purely local, and improvement in one direction may prevent greater progress in the other direction. Furthermore, the layout may be blocked in both dimensions, but still be far away from an

optimal solution. An overview of one-dimensional compaction techniques for VLSI can be found in (Lengauer, 1990) and (LaPaugh, 1998).

Algorithm 4.3 shows a generic improvement heuristics in orthogonal graph drawing which is similar to the one-dimensional compaction schemes known from VLSI. It repeatedly performs one-dimensional compaction steps until no further improvement is possible. The compaction steps in lines 4 and 5 refer to the one-dimensional problems in Section 4.1, and the improvement heuristics presented in this section differ in how they perform these steps.[5] We will present heuristics that solve in particular problems W-ICOMP, H-ICOMP, and L-ICOMP to optimality and describe two different approaches: First, we introduce the compression ridge-based algorithms. These heuristics look for cuts through the drawing that represent unnecessarily long edges. The second approach makes use of special constraint graphs and is known as graph-based compaction in VLSI design. We show how to exploit the visibility properties of an existing drawing in order to create a pair of so-called *visibility graphs* which can be used to produce a better drawing,

Algorithm 4.3 *One-dimensional compaction scheme*

Input: Orthogonal grid drawing Γ for orthogonal representation H

Output: Orthogonal grid drawing Γ' for H

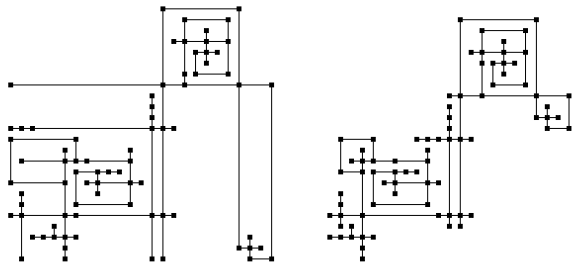
1: $\Gamma' = \Gamma$;
2: **repeat**
3: $\quad L_{\text{old}} = L(\Gamma')$;
4: $\quad$ perform one-dimensional compaction step in x-direction on Γ';
5: $\quad$ perform one-dimensional compaction step in y-direction on Γ';
6: $\quad L_{\text{new}} = L(\Gamma')$;
7: **until** $L_{\text{old}} \equiv L_{\text{new}}$; // no further improvement

Remark 4.5. Before presenting the different improvement heuristics, we want to make a general statement about the above one-dimensional compaction scheme: Even if area minimization is the ultimate goal, it is in general better—but also more difficult—to solve one-dimensional subproblems of type L-ICOMP instead of W-ICOMP or H-ICOMP in lines 4 and 5. Reducing width and height mainly affects the boundary of the drawing, whereas improving the edge length additionally yields several local improvements inside the drawing which may unblock parts of the opposite direction and may allow a greater progress in the succeeding step. Figure 4.14 illustrates this behavior with the example of Figures 4.10 and 4.13.

Compression Ridge-Based Heuristics

In VLSI, a *compression ridge* refers to a cut that divides the layout into two parts and passes through regions of empty space. If such a cut is found, the parts can be pushed together as much as possible without violating any of the spacing constraints. Depending on the

[5] Lines 4 and 5 may be swapped, of course, algorithms will then start with a vertical compaction step.



(a) (b)

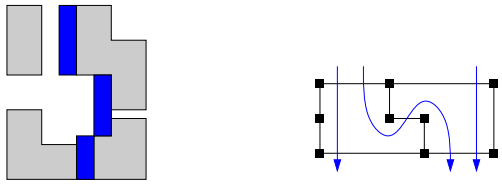
Figure 4.14: Optimization goal in each iteration: (a) One-dimensional width and height. (b) One-dimensional total edge length

direction of the compression ridge (top-to-bottom or right-to-left), this corresponds to a horizontal or vertical compaction step. Akers *et al.* (1970) first mention this method in the context of virtual grid compaction.

We present the method of Dai and Kuh (1987) and adapt it to fit into the one-dimensional compaction scheme for orthogonal grid drawings (Algorithm 4.3). The regions of empty space in a VLSI symbolic layout correspond to edges that are longer than the minimum length of one unit. After identifying a compression ridge in a drawing for an orthogonal representation H, the algorithm decreases the lengths of the edges that are cut by the ridge by the maximum possible amount: The shortest edge e_s on the ridge determines this value; the length of every edge on the cut can decrease by the length of e_s minus one. The result is again an orthogonal grid drawing for H with reduced horizontal or vertical edge length. In some cases it is advantageous to increase the lengths of a few edges in order to allow more improvement for other edges. In these cases, the ridge runs temporarily in the opposite direction. Figure 4.15 illustrates the method; the middle ridge in Figure 4.15(b) indicates that the length of one edge must increase by one unit so that two other edge lengths can decrease.

As Figure 4.15 indicates, several ridges in one direction can exist simultaneously. Disjunct ridges can be combined and the resulting compaction step corresponds to line 4 or line 5 in the one-dimensional compaction scheme. To find the best ridges in terms of one-dimensional compaction progress, Dai and Kuh (1987) introduce the *tile graph*, a network that administrates the regions of empty space. In graph drawing the tile graph corresponds to a dual graph that depends on the initial drawing. Each flow in the network corresponds to a set of ridges, its value is equal to the possible decrease in edge length. An interpretation of a maximum flow in the network corresponds to the best cuts. Thus, the compression ridge method solves the one-dimensional compaction L-ICOMP to optimality.

Figure 4.16 illustrates the construction of the horizontal network N_x, analogous rules apply for the vertical tile graph. As a first step, a compression ridge-based algorithm dis-



(a) VLSI (b) Orthogonal graph drawing

Figure 4.15: Compression ridges in (a) VLSI and (b) orthogonal graph drawing. In (a), the ridge is the darker part, in (b) three ridges correspond to the three arrows

sects the initial drawing into horizontal stripes, see Figure 4.16(a); the process divides the faces in the drawing into rectangles.[6] Each rectangular face f in the drawing corresponds to a node $n(f)$ in network N_x. In addition, there are two nodes, s and t for the external face; s at the top of the drawing and t at the bottom, see Figure 4.16(b). Arcs are directed downwards: For each horizontal edge e separating an upper face f from a lower face g, there is an arc $a_e^+ = (n(f), n(g))$ and an arc $a_e^- = (n(g), n(f))$. The capacity of a_e^+ is the length of e minus one and corresponds to the maximal possible decrease of the length of e. The opposite arc a_e^- has infinite capacity, accounting for possible elongations of e. Figure 4.16(b) shows the network for the example.

The maximum flow from s to t in this network corresponds to an optimal solution of problem W-ICOMP which asks for minimum one-dimensional width, see also Definition 2.6 on page 23. Figure 4.16(c) and (d) illustrates these steps in the example. Each compaction step has running time $O(n \log n)$ for an initial drawing with n vertices. The bottleneck is the computation of a maximum flow problem in N_x. Since network N_x is the dual of the underlying orthogonal representation it is planar and linear in size of the input. The maximum s-t flow in a planar network can be computed in time $O(n \log n)$, see, *e.g.*, (Ahuja *et al.*, 1993).

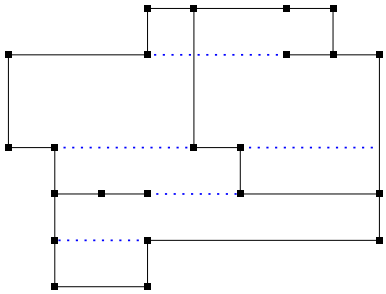
This gives rise to two compression ridge-based improvement heuristics for the two-dimensional compaction problems in graph drawing that fit into the one-dimensional compaction scheme. As the above discussion shows, computing a maximum flow in the network solves problem W-ICOMP to optimality; the same holds, of course, for H-ICOMP during the vertical compaction step. A second heuristics is faster, but does not result in optimal solutions for W-ICOMP or H-ICOMP: A linear time depth-first search in the network without upward arcs will also identify a ridge—though not necessarily a good one. Adapting the network and repeating the search results in a set of cuts for each direction.

In graph drawing, the 4M-algorithm by Fößmeier *et al.* (1998) uses the compression ridge method as one of its post-processing steps. See also Chapter 7 for a discussion of the interaction of compaction and post-processing.

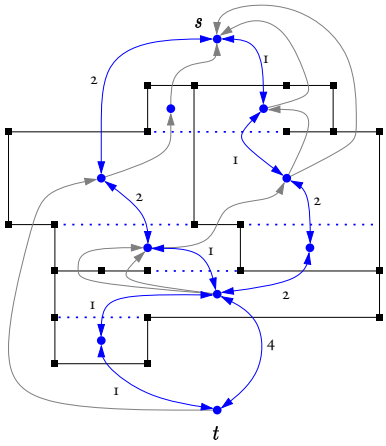
[6] There is a close relationship to the rectangular dissection method of Section 4.2.4. In (Mühlenbacher, 2001), the dissection in horizontal or vertical stripes takes place at the level of the orthogonal representation, resulting in a compaction algorithm which combines elements of constructive and improvement heuristics.

Constraint Graph-Based Heuristics

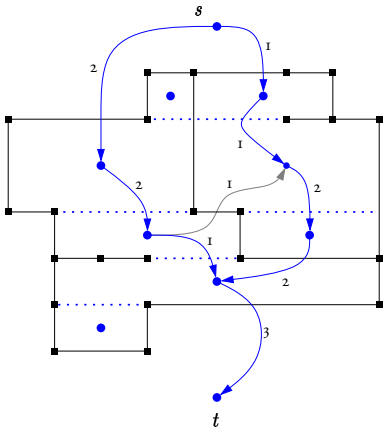
In logical circuit design, the compression ridge-based techniques have the disadvantage that the initial layout must be legal to apply them. More importantly, their running time is not acceptable for a repeated use in Algorithm 4.3 in the case of very large instances as they occur in chip design. Faster and closer to the design rules are the constraint graph-based algorithms, that appear first in (Williams, 1978) and (Dunlop, 1978) and in the area



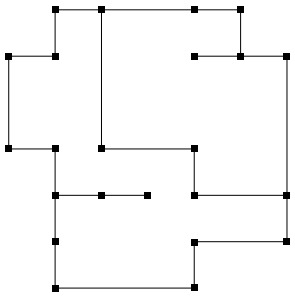
(a) Horizontal stripes (shown as dashed lines) dissect the initial drawing



(b) Network N_x with capacities; the grey upward arcs have infinite capacity



(c) Maximal s-t-flow in N_x; only arcs with flow are shown



(d) Resulting drawing after changing the edge lengths

Figure 4.16: The compression-ridge method in graph drawing: horizontal compaction step

of symbolic layout in (Hsueh, 1979). In this section, we present two constraint graph-based improvement heuristics for orthogonal graph drawing that fit into the scheme of Algorithm 4.3. Consider the one-dimensional compaction problems which appear within the scheme. We will prove that the first heuristics solves subproblems W-ICOMP and H-ICOMP to optimality and that the second heuristics provides optimal solutions for L-ICOMP.

In order to model the relative placements of segments in an orthogonal grid drawing, we define the *visibility graphs* as a special pair of placement graphs. In a drawing Γ, the placement of vertices and bends is fixed. Informally, the visibility graphs reflect the positioning relations of all pairs of segments that are intersected by a common horizontal or vertical line, possibly via the transitivity of the arc sets.

Two vertical segments I and J are *left-right-visible* in Γ if I is to the left of J and there is a virtual horizontal line segment from I to J that does not intersect another vertical segment. Likewise, we define *bottom-top-visibility* for horizontal segments. The following is a precise definition of visibility graphs:

Definition 4.10 (Visibility graphs). Let $\Gamma = (\Gamma_x, \Gamma_y)$ be a grid drawing of an orthogonal representation H of a four-planar graph $G = (V, E)$. A pair of *visibility graphs* $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ is a pair of placement graphs, *i.e.*, $N_x = \mathcal{S}_v$, $N_y = \mathcal{S}_h$. All arcs have unit weight[7] and result from visibility properties of the drawing:

$$A_x = \{(I, J) \in N_x \times N_x \mid I \text{ and } J \text{ left-right-visible}\}$$
$$A_y = \{(I, J) \in N_y \times N_y \mid I \text{ and } J \text{ bottom-top-visible}\} \ .$$

Figure 4.17 shows a pair of visibility graphs. As for the level of the shape, on which an orthogonal representation H gives rise to a unique pair of shape graphs $\sigma(H)$, every drawing of H gives rise to a unique pair of visibility graphs that we will refer to by $\nu(\Gamma)$.

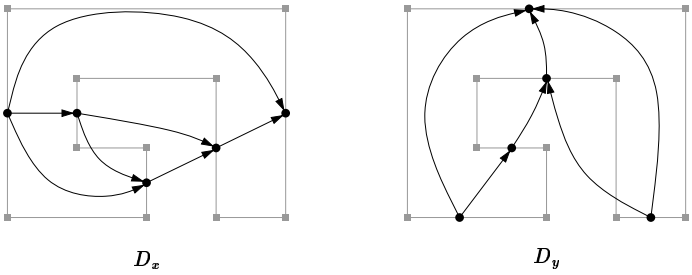


Figure 4.17: Visibility graphs $\nu(\Gamma) = (D_x, D_y)$ for an orthogonal drawing Γ

Visibility graphs have a close relationship to the *layout graphs* used in symbolic circuit layout. The layout graphs result from the design rules and contain the necessary arcs to separate the components. A crucial issue in VLSI design is to construct the arc sets in the

[7] As for the shape graphs, more complex variants of visibility graphs exist by allowing general weights on the edges. For the sake of simplicity we assume unit weights, see also Remark 4.1 on page 46.

layout graphs so that their transitive closures guarantee a feasible feature placement. On the one hand, it is desirable to create the smallest arc sets with this property—these are the *transitive reductions*—in order to speed up the computation of assignments. On the other hand, the construction itself should be efficient. In VLSI design, the latter issue is even more important than in orthogonal graph drawing due to the large sizes of instances, while the first issue is more difficult because of the different kinds of constraints. Schlag, Luccio, Maestrini, Lee, and Wong (1984) study the complexity of this problem and consider various special cases. Doenhardt and Lengauer (1987) present algorithms to compute the transitive reductions efficiently.

The following observation shows that visibility graphs can be seen as extensions of shape graphs. This also motivates the approach in the next section where we will characterize the space of complete extensions of shape graphs—in a way this is the space of possible visibility graphs for a given representation.

Observation 4.1. *Let Γ be a drawing of a simple orthogonal representation H. The arc sets of the visibility graphs $\nu(\Gamma)$ contain the arc sets of shape graphs $\sigma(H)$.*

Proof. In the horizontal shape graph there exists an arc $a = (I, J)$ for each horizontal half-edge $(v, w) \in \vec{E}_r$. Clearly, in any drawing Γ for H the segments I and J are left-right-visible, since $\Gamma\big((v, w)\big)$ is a line segment between I and J that obviously does not intersect any other segment. Hence a belongs also to the horizontal visibility graph. Analogous considerations apply to the vertical arc set. ∎

Observation 4.1 allows us to use visibility graphs in order to compute feasible assignments for shape graphs.

Lemma 4.6. *Let Γ be a drawing for a simple orthogonal representation H. A feasible assignment for the pair of visibility graphs $\nu(\Gamma)$ is also feasible for the underlying shape graphs $\sigma(H)$.*

Proof. Removing arcs from a constraint graph for which a feasible assignment exists does not destroy the feasibility of the assignment. According to Observation 4.1, we can obtain the shape graphs by removing a set of arcs from the visibility graphs. ∎

Lemma 4.6 encourages us to use visibility graphs inside the one-dimensional compaction scheme (Algorithm 4.3 on page 62). We replace line 4 in the scheme by

```
4a: construct visibility graph D_x;
4b: compute minimal or Δ-minimal assignment for D_x;
4c: assign new x-coordinates;
```

and replace line 5 by

```
5a: construct visibility graph D_y;
5b: compute minimal or Δ-minimal assignment for D_y;
5c: assign new y-coordinates;
```

.

It remains to be proven, however, that this replacement results in a correct algorithm. Furthermore, we have to specify how to construct the visibility graphs in lines 4a and 5a.

We will first show that a feasible assignment for one of the two visibility graphs results in a feasible solution of the two-dimensional compaction problems and will then present an efficient algorithm to build the visibility graphs.

As a first step towards a correctness proof for the constraint graph-based improvement heuristics, we prove that visibility graphs, like shape graphs, are upward planar.

Lemma 4.7. *Visibility graphs are upward planar.*

Proof. By construction, the graphs are upward and Definition 4.10 provides an upward planar embedding (see also Figure 4.17): Visibility expresses the fact that a virtual horizontal or vertical line from one segment to the other exists that does not cross a third segment. Imagine that the arcs run along these virtual lines—by definition they do not cross—which results in an upward planar embedding. ∎

Lemma 4.8. *Let $\Gamma = (\Gamma_x, \Gamma_y)$ be a drawing of an orthogonal representation H and let $\nu(\Gamma) = (D_x, D_y)$ be the corresponding visibility graphs. A feasible assignment for D_x results in an orthogonal grid drawing (Γ'_x, Γ_y) for H, a feasible assignment for D_y results in an orthogonal grid drawing (Γ_x, Γ'_y) for H.*

Proof. Two segments I and J in Γ *overlap horizontally* if the intervals formed by their x-coordinates intersect. Similarly, I and J *overlap vertically* if the intersection of y-intervals is non-empty.

We consider the process of changing the x-coordinates via a computation of a feasible assignment for the horizontal visibility graph D_x. Assume the resulting placement violates Definition 2.1, *i.e.*, it does not result in a correct orthogonal grid drawing. The violation must be due to vertically overlapping segments, since the y-coordinates do not change during the process. By definition, the horizontal visibility graph protects the relative positioning constraints between any pair of vertically overlapping segments. This is a contradiction to the feasibility of the assignment.

Furthermore, the resulting drawing respects the shape of the orthogonal representation H: By Lemma 4.6 the assignment is also feasible for the underlying shape graphs. ∎

Remark 4.6. Visibility graphs are not necessarily complete, but in some sense almost complete. For complete pairs of placement graphs, any coordinate assignment (c_x, c_y) leads to feasible solutions for the compaction problems, whereas visibility graphs allow the recomputation of only one of the assignments: either c_x and c_y may be changed, but not both.

Depending on the usage of minimal or Δ-minimal assignments in lines 4a and 5a, this strategy results in two constraint graph-based improvement heuristics for the two-dimensional compaction problems. Computing minimal assignments leads to optimal solutions of the one-dimensional compaction problems W-ICOMP and H-ICOMP in each step. Using the flow-based methods that yield Δ-minimal assignments results in optimal solutions for problem L-ICOMP.

We use an efficient scan line algorithm that is similar to the one presented in (Schlag *et al.*, 1984) to compute the arc sets of the visibility graphs. We restrict our description to the construction of the arc set A_x, the algorithm for the vertical arc set is similar.

We represent each vertical segment S by a triple (x_s, b_s, t_s) where x_s is the x-coordinate of S, b_s its bottom y-coordinate, and t_s its top y-coordinate. Then we scan the segments from top to bottom. During the scan, a height-balanced tree holds the segments that cross the horizontal scan line in their left-to-right order. Once the scan line encounters a top end t_s of a segment S the algorithm inserts S into the height-balanced tree according to its x-coordinate x_s and looks for the left and right neighbors of S in the tree. Let L and R denote the left and right neighbors of a segment S, respectively. If there is a left neighbor L of S, the algorithm adds the arc (L, S) to A_x, if there is a right neighbor R, it adds the arc (S, R). If the scan line encounters a bottom end b_s of a segment S, the algorithm deletes S from the tree and adds an arc from L to R if both neighbors are present.

This method constructs the visibility graphs efficiently:[8]

Lemma 4.9. *Let Γ be a drawing of an orthogonal representation H with n vertices. The algorithm constructs the pair of visibility graphs $\nu(\Gamma)$ in time $O(n \log n)$.*

Proof. By Lemma 4.1, the number of segments is $O(n)$. The algorithm inserts and deletes each segment once from the height-balanced tree; both operations can be done in time $O(\log n)$. The algorithm is correct since the scan line is exactly the virtual line between the segments from the definition of left-right- and bottom-top-visibility: During the construction of the horizontal arc set, the height-balanced tree contains at each point of time only vertically overlapping segments. Since the segments are sorted according to their x-coordinates, the algorithm inserts only arcs between visible segments. The proof for the vertical direction is similar. ∎

We analyze the running time of an iteration in Algorithm 4.3 using visibility graphs and the above algorithm to construct them.

Lemma 4.10. *Let Γ be a drawing of an orthogonal representation H with n vertices. An iteration in the one-dimensional compaction scheme (Algorithm 4.3) takes time*

(a) $O(n \log n)$ *when computing minimal assignments.*
(b) $O(n^2 (\log n)^2)$ *when computing Δ-minimal assignments.*

Proof. The running time of one iteration is determined by the time needed for lines 4 and 5. Using constraint graph-based heuristics with visibility graphs we have to consider lines 4a-4c and 5a-5c. Lemma 4.9 shows that lines 4a and 5a take time $O(n \log n)$ each. Lines 4c and 5c clearly can be done in linear time. Since the visibility graphs are upward planar, the number of nodes in both graphs is $O(n)$ and all arc weights are non-negative, lines 4b and 5b take time $O(n)$ when computing a minimal assignment with Algorithm 3.2 and time $O(n^2 (\log n)^2)$ when computing a Δ-minimal assignment with Algorithm 3.3. ∎

[8] Schlag *et al.* (1984) remark that a theoretical speed-up to $O(n \log \log n)$ is possible by using the data structure defined in (van Emde Boas, 1977)

Analysis of the One-Dimensional Compaction Scheme

It is an interesting question how many iterations the one-dimensional compaction scheme performs until both directions are blocked and no further one-dimensional improvement is possible. The next lemma shows that instances exist for which a linear number of steps is necessary.

Lemma 4.11. *There is an orthogonal grid drawing Γ of a graph with n vertices for which the one-dimensional compaction scheme needs $O(n)$ steps.*

Proof. Consider the drawing of the graph in Figure 4.18(a). It consists of an even number of U-shaped components each of which contains four vertices and may be turned by 180 or 270 degrees; the example shows eight components. The first component is not turned and shown in grey color. The only one-dimensional improvement consists of shrinking the longer vertical segment in the first component, resulting in Figure 4.18(b). Figures 4.18(c) and 4.18(d) show the drawing after steps two and three. In each step only one component shrinks and unblocks the next component that is of opposite direction. Clearly, the number of iterations I until the process stops with Figure 4.18(f) is half the number of U-shaped components, more precisely:

$$\begin{aligned} I &= \frac{n}{4}/2 \\ &= O(n) \quad . \end{aligned}$$

Thus, for certain instances $O(n)$ one-dimensional steps are needed until both directions are blocked and the one-dimensional compaction scheme terminates. ∎

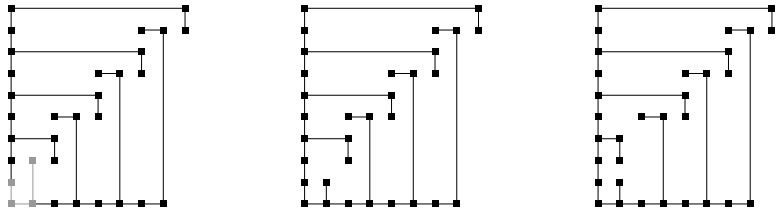
In examples occurring in practice, however, this bound is almost never met. Our computational experiments in Section 4.4 also contain the numbers of iterations the different strategies need for each graph in three test-suites in experimental graph drawing; see Table 4.4 on page 109. The maximal number of iterations measured is seven among more than 12,500 graphs of up to 3,500 vertices.

To conclude the section on heuristic methods for compaction problems, we show that algorithms within the one-dimensional compaction scheme 4.3 do not constitute constant factor approximation algorithms for the two-dimensional compaction problems.

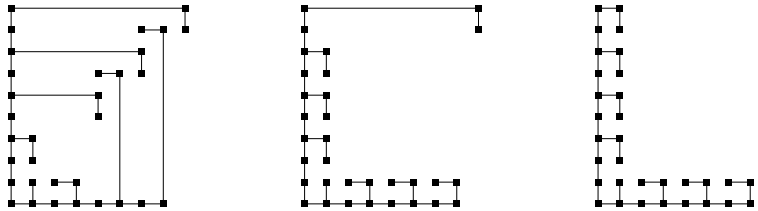
Lemma 4.12. *Algorithms in the one-dimensional compaction scheme 4.3 do not approximate L-COMP and l_{max}-COMP within a constant factor.*

Proof. Consider the constructions in Figure 4.19. They consist of k consecutive horizontal edges and ℓ L-shaped components that each contain two edges. Figure 4.19(a) shows such a construction for $\ell = 2$. The total edge length of the drawing is $2k + 5$ and cannot be improved by the one-dimensional compaction scheme, since both directions are blocked. The drawing in Figure 4.19(b) is optimal with respect to total edge length that amounts to $k + 6$. Figures 4.19(a) and (b) show the construction for $\ell = 5$. Here the edge lengths are $5k + 20$ and $k + 30$, respectively.

We make similar observations concerning the maximum edge length in the drawings: In Figures 4.19(a) and (c), the maximum edge length is $k + 1$ and $k + 4$, respectively. In



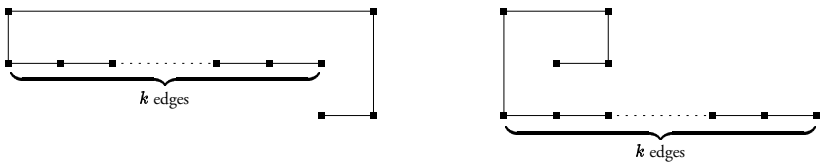
(a) Initial drawing (b) After first step (c) After second step



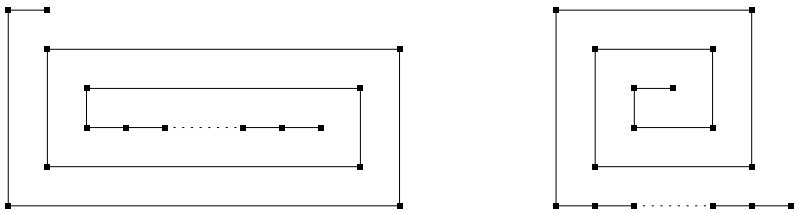
(d) After third step (e) After last but one step (f) After last step

Figure 4.18: Grid drawing requiring many one-dimensional iterations

the optimal drawings in Figures 4.19(b) and (d), the lengths of the longest edges are 2 and 5.



(a) $\ell = 2, L = 2k + 5, l_{\max} = k + 1$ (b) $\ell = 2, L = k + 6, l_{\max} = 2$



(c) $\ell = 5, L = 5k + 20, l_{\max} = k + 4$ (d) $\ell = 5, L = k + 30, l_{\max} = 5$

Figure 4.19: A drawing generated with (a), (c) an iterative one-dimensional and (b), (d) an optimal compaction method.

For general ℓ the following two formulas for the total edge lengths in the initial drawings (which do not allow one-dimensional improvement) and the optimal drawings are easily proved by induction:

$$\ell k + 5(\ell - 1) \qquad \text{and} \qquad k + \sum_{1}^{\ell} 2i \ .$$

The maximum edge lengths are

$$k + \lfloor \ell/2 \rfloor \qquad \text{and} \qquad \ell \ ,$$

respectively.

We choose $k = \ell^2$. Let α_L be the approximation factor for L-COMP and let $\alpha_{\max}$ be the approximation factor for $l_{\max}$-COMP. The choice of k results in the factors

$$\alpha_L = \frac{\ell^3 + 5(\ell-1)}{\ell^2 + 2\ell(\ell-1)} = \frac{1}{3}\ell - \frac{2}{9} + \frac{\frac{41}{9}\ell - 5}{3\ell^2 - 2\ell}$$
$$\alpha_{\max} = \frac{\ell^2 + \lfloor \frac{\ell}{2} \rfloor}{\ell} > \frac{\ell^2 + \frac{\ell}{2} - 1}{\ell} = \ell + \frac{1}{2} - \frac{1}{\ell} \ .$$

Increasing ℓ also increases α_L and $\alpha_{\max}$, *i.e.*,

$$\lim_{\ell \to \infty} \alpha_L = \infty \quad \text{and} \quad \lim_{\ell \to \infty} \alpha_{\max} = \infty \ ,$$

which demonstrates that no constant approximation factor exists. ∎

The examples in Figure 4.19 show that the one-dimensional compaction methods fail if both directions are blocked. If a blocking situation occurs early in the compaction process, the one-dimensional methods perform poorly.

4.2.6 Compaction Reformulated

In this section we present a combinatorial formulation of the two-dimensional compaction problems that is based on the underlying shape graphs of an instance H. We define *extensions* of shape graphs and show that the space of complete extensions is in one-to-one correspondence to the space of orthogonal grid drawings of H.

The previous section already uses extensions of shape graphs: the visibility graphs. However, as shown in the analysis of the one-dimensional compaction scheme, these extensions are often too strict, since they reflect the one-dimensional visibility properties between the segments in an already existing drawing of the graphs. Visibility is a property of the drawing and not of the underlying instance of the appropriate compaction problem—if the drawing is bad, a compaction algorithm must adhere to the unfortunate initial placement decisions. To obtain an optimal drawing it is often necessary to ignore certain of the visibility properties and to replace them by separation constraints in the opposite direction.

On the other hand, shape graphs are generally not strict enough. Clearly, the information coded in the shape graphs must be respected in any drawing with the appropriate shape. But simply computing a coordinate assignment for the shape graphs is in general not successful: In most cases, this strategy does not result in an orthogonal drawing but in an infeasible placement with crossing edges and vertices placed on top of other vertices as, for instance, in Figure 4.20. Though not feasible, the values for area as well as for total and maximum edge length constitute lower bounds for the respective values in feasible drawings.

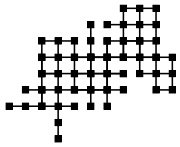


Figure 4.20: Information about the shape is not enough: Infeasible placement of vertices (same shape as drawings in Figures 4.10, 4.13 and 4.14). Coordinates result from Δ-minimal assignment for the shape graphs

The reason for the infeasibility of the resulting placement lies in the fact that the underlying shape graphs D_x and D_y are in general not complete. In order to solve the two-dimensional compaction problems, our idea is to look for an extension of D_x and D_y; in some sense we want to find a perfect pair of visibility graphs for which the shape graphs are subgraphs. In this section we reformulate the compaction problems of Section 4.1 as combinatorial problems in so-called *extensions* of D_x and D_y.

Definition 4.11 (Extension). Let H be an orthogonal representation and let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be the corresponding pair of shape graphs $\sigma(H)$. An *extension* of D_x and D_y is a pair of placement graphs $\widehat{D}_x = (N_x, \widehat{A}_x, \widehat{w}_x)$ and $\widehat{D}_y = (N_y, \widehat{A}_y, \widehat{w}_y)$ with

$$A_x \subseteq \widehat{A}_x \quad , \quad A_y \subseteq \widehat{A}_y$$

and weights $\widehat{w}_x(a) = w_x(a)$ for all $a \in A_x$ and $\widehat{w}_y(a) = w_y(a)$ for all $a \in A_y$.

Among the extensions of $\sigma(H)$ we are especially interested in those satisfying the property of completeness, *i.e.*, the *complete extensions* of the shape graphs. We know by Theorem 4.1 that we can use a complete pair of placement graphs to produce an orthogonal drawing for the underlying graph by computing a feasible assignment for each constraint graph. By Definition 4.11 and with the same argument as for visibility graphs (Lemma 4.6), a feasible assignment for an extension is also feasible for the shape graphs. Hence, a drawing obtained via computing a coordinate assignment for complete extensions of $\sigma(H)$ is a drawing of H.

Definition 4.12 (Compaction problems: combinatorial formulation). Given an orthogonal representation H, find a complete extension $(\widehat{D}_x, \widehat{D}_y)$ of the shape graphs $\sigma(H) = (D_x, D_y)$ and a coordinate assignment for this extension that minimizes

(a) the minimum total edge length $L(\Gamma)$
(b) the maximum edge length $l_{\max}(\Gamma)$
(c) the area $A(\Gamma)$

of a corresponding drawing Γ.

The following theorem characterizes the set of feasible solutions for the compaction problems and shows the equivalence between the combinatorial formulations of Definition 4.12 and the original formulations of Definition 4.1.

Theorem 4.4. *Let H be an orthogonal representation and let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be the corresponding shape graphs $\sigma(H)$.*

Every simple orthogonal grid drawing of H corresponds to a coordinate assignment (c_x, c_y) for a complete extension $(\widehat{D}_x, \widehat{D}_y)$ of $\sigma(H)$ and vice versa: every coordinate assignment (c_x, c_y) for a complete extension $(\widehat{D}_x, \widehat{D}_y)$ of the shape graphs results in a simple orthogonal grid drawing of H.

Proof. To prove the first part of the theorem, we consider an orthogonal grid drawing $\Gamma = (\Gamma_x, \Gamma_y)$ of H. We construct a pair of placement graphs $\widehat{D}_x = (N_x, \widehat{A}_x, \widehat{w}_x)$ and $\widehat{D}_y = (N_y, \widehat{A}_y, \widehat{w}_y)$ and two assignments c_x and c_y as follows.

First, we define the assignments using the coordinates in Γ:[9]

$$\begin{aligned} c_x(I) &= \Gamma_x(v) \quad \text{for some } v \in I \text{ for all } I \in N_x \\ c_y(J) &= \Gamma_y(v) \quad \text{for some } v \in J \text{ for all } J \in N_y \end{aligned}$$

and then the arcs and their weights through the assignments:

$$\begin{aligned} \widehat{A}_x &= \{(I, J) \in N_x \times N_x \mid c_x(I) < c_x(J)\} \\ \widehat{A}_y &= \{(I, J) \in N_y \times N_y \mid c_y(I) < c_y(J)\} \\ w_x(a) &= 1 \quad \text{for all } a = (I, J) \in \widehat{A}_x \\ w_y(a) &= 1 \quad \text{for all } a = (I, J) \in \widehat{A}_y \ . \end{aligned}$$

Note that the construction of the arc sets is similar as in the definition of visibility graphs as given on page 66. We verify the following properties and thus conclude the proof of the first part of Theorem 4.4.

(a) The pair of placement graphs $(\widehat{D}_x, \widehat{D}_y)$ is an extension of the shape graphs.
The argument is similar as for visibility graphs: Consider an arc $a = (I, J) \in A_x$. It corresponds to a horizontal half-edge $(v, w) \in \vec{E}_r$ that is represented by a line segment from $\Gamma(v)$ to $\Gamma(w)$. The construction of the assignments ensures $c_x(J) > c_x(I)$, it follows that $a \in \widehat{A}_x$. Analogous considerations hold for vertical arcs.

(b) The extension $(\widehat{D}_x, \widehat{D}_y)$ is complete.
We show first that the extension does not contain cycles: Assume there is a cycle

[9] Observe that $\Gamma_x(v) = \Gamma_x(w)$ for all $v, w \in I \in N_x$ and $\Gamma_y(v) = \Gamma_y(w)$ for all $v, w \in J \in N_y$.

$C = \big((S_1, S_2), (S_2, S_3), \ldots, (S_{k-1}, S_k), (S_k, S_1)\big)$ in one of the arc sets $\widehat{A}_x$ or $\widehat{A}_y$. The constraints coded in C express the contradiction that S_1 is either to the left of itself, if $C \subseteq \widehat{A}_x$, or below itself, if $C \subseteq \widehat{A}_y$.

Assume now that two segments I and J are not separated, *i.e.*, none of the four paths $r_I \xrightarrow{*} l_J$, $r_J \xrightarrow{*} l_I$, $t_I \xrightarrow{*} b_J$, and $t_J \xrightarrow{*} b_I$ exists.

For the transitive closures of the arc sets this means

$$\big(r_I, l_J\big) \notin \widehat{A}_x^* \qquad \big(r_J, l_I\big) \notin \widehat{A}_x^* \tag{4.3}$$

$$\big(t_I, b_J\big) \notin \widehat{A}_y^* \qquad \big(t_J, b_I\big) \notin \widehat{A}_y^* \ . \tag{4.4}$$

By (4.3) and by definition of the arc sets we get $x(v) = x(w)$ for all $v \in I$ and $w \in J$. Similarly, (4.4) requires $y(v) = y(w)$ for all $v \in I$ and $w \in J$. It follows $I = J$, a contradiction, since a segment is separated from itself by definition.

(c) The pair of assignments (c_x, c_y) is a coordinate assignment for $(\widehat{D}_x, \widehat{D}_y)$.
Both assignments are feasible by construction. We show that c_x is feasible for $\widehat{D}_x$, the proof for c_y and D_y is similar. For each arc $a = (I, J) \in \widehat{A}_x$ the following inequality must be valid.

$$c_x(J) - c_x(I) \geq w(a) = 1$$

which trivially holds since Γ is a grid drawing.

The backward direction follows directly by Theorem 4.1 and the fact that feasible assignments for extensions are also feasible for subgraphs of extensions (Observation 3.1).

■

Definition 4.12 provides a graph-theoretical translation of the originally geometric compaction problems of Section 4.1. The new task is to find a complete extension of the given shape graphs and an assignment for the extension that optimizes one of the aesthetic criteria. An interesting class of shape graphs are those for which the extension is uniquely determined. This situation occurs when all pairs of segments are either separated or only one possibility of separating them without creating a cycle exists. We provide a general definition for placement graphs:

Definition 4.13 (Uniquely completable). A pair of placement graphs is *uniquely completable* if and only if

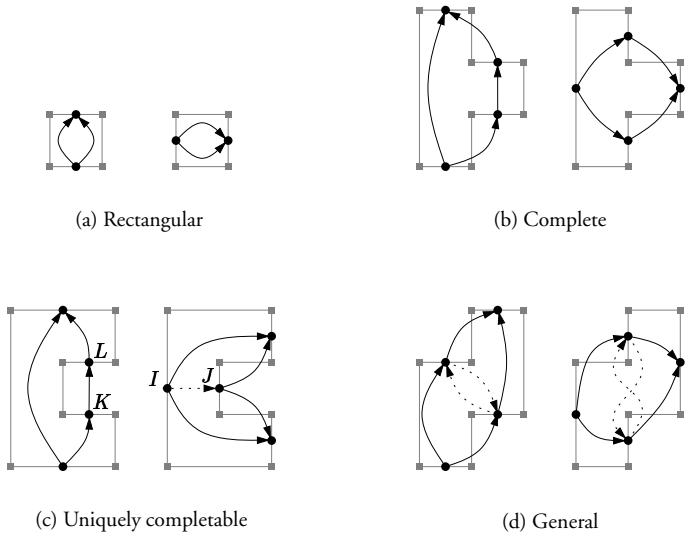
(a) Both graphs are acyclic;

(b) All pairs of segments (I, J) are either separated or three of the four paths $r_I \xrightarrow{*} l_J$, $r_J \xrightarrow{*} l_I$, $t_I \xrightarrow{*} b_J$ and $t_J \xrightarrow{*} b_I$ induce a cycle.

An orthogonal representation H is *uniquely completable* if and only if the corresponding shape graphs $\sigma(H)$ are uniquely completable.

Clearly, a complete pair of placement graphs is also uniquely completable according to this definition. Figure 4.21 shows four different kinds of representations with their shape graphs. The representation in Figure 4.21(c) is uniquely completable since its only unseparated segment pairs of opposite direction are (I, K) and (I, L). Evaluating the four possible paths in the definition of completeness shows that in each case only one of the paths does not induce a cycle: For the segments I and K, this is $r_I \xrightarrow{*} l_K$, and for the segments I and L, the path is $r_I \xrightarrow{*} l_L$. The situation is different in Figure 4.21(d): Here, none of the four paths introduces a cycle for several pairs of unseparated segments and it is not clear how to extend the shape graphs since there are many different possibilities; it is the combination of these choices that makes the compaction problems hard at the combinatorial level.

The definition of unique completability refines the hierarchy of orthogonal representations. Figure 4.22 illustrates the inclusion relations between classes of representations.



(a) Rectangular (b) Complete

(c) Uniquely completable (d) General

Figure 4.21: Orthogonal representations and their shape graphs. Each example is a member of the respective class but not of its preceding classes

Unfortunately, the most frequent type of representation in practice is the general type. We can use the concept of unique completability, however, to insert as many arcs as possible into the shape graphs. This will be exploited in the next section. In general, only a few segment pairs are the reason that a representation is not uniquely completable, and many relations can be established according to Definition 4.13: If only one of the four possible paths does not induce a cycle, we can add the corresponding arc to the shape graphs that uniquely separates the segments.

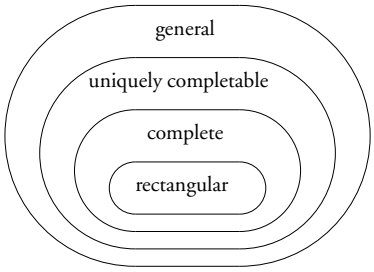


Figure 4.22: Inclusion hierarchy of classes of orthogonal representations

4.3 Exact Compaction Algorithms

In this section we present exact algorithms for the compaction problems, *i.e.*, methods that always compute an optimal solution. We exploit combinatorial properties of the different classes of orthogonal representations presented in the hierarchy in Figure 4.22. Section 4.3.1 contains an exact polynomial-time algorithm for representations in all but the last layer of Figure 4.22. For general orthogonal representations we develop integer linear programming formulations in Section 4.3.2 that correspond to the combinatorial equivalents of the compaction problems. The ILP formulations enable us to use the generic branch-and-bound and branch-and-cut algorithms presented in Chapter 2. We describe algorithms that solve the compaction problems that ask for minimum total edge length, width, height, half-perimeter, and maximum edge length to optimality. The ILP formulations give rise to exact LP-based algorithms: Section 4.3.3 describes a branch-and-bound algorithm. An extension of this algorithm by adding cutting planes results in a branch-and-cut algorithm.

Although these branch-and-bound and branch-and-cut algorithms need exponential time in the worst case, our experimental study in Section 4.4 shows that they are suitable for practical applications: an implementation of the branch-and-bound algorithm solves all graphs in a large test-suite of practical graphs in short computation time.

4.3.1 Compacting Uniquely Completable Representations

We present an algorithm that computes a maximal unique completion of a given pair of placement graphs, *i.e.*, an extension that cannot be completed further according to Definition 4.13. In particular, the algorithm outputs the pairs of segments that have to be separated in order to obtain a pair of complete placement graphs. In the cases of complete or uniquely completable placement graphs, this set is empty and the extension is complete; in this case we are able to apply polynomial-time algorithms to solve the two-dimensional compaction problems to optimality.

Again, we exploit Lemma 4.4 and Corollary 4.3 in the algorithm: we only need to ensure local completeness in each face and restrict the focus to segments of opposite direction. But simply checking for each pair of opposite segments in each face whether they

satisfy the property in Definition 4.13(b) does not suffice. A unique separation for one pair could have further consequences for other pairs. Inserting a new arc may separate previously non-separated pairs or may cause other pairs to be uniquely separable.

We will therefore maintain the transitive closure of the arc set in each face and update it appropriately. Let H be an orthogonal representation and let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be a pair of placement graphs for H. Let $V(f)$ be the vertices on the boundary of a face f of H and let

$$A(f) = \{(I, J) \in A_x \cup A_y \mid \exists v \in I \cap V(f) \text{ and } \exists v \in J \cap V(f)\}$$

be the *local arc set* of the face. Roughly speaking, a local arc set contains those arcs of the placement graphs for which both endpoints belong to the same face. We will represent the transitive closure $A^*(f)$ of a local arc set $A(f)$ by a bit array that enables us to test for the presence of edges and paths in constant time.

We will need the following procedure as a subroutine and refer to it as *separated*(I, J). It checks for two segments I and J in the same face whether the appropriate arc is contained in the transitive closure of the local arc set. If this is the case, the two segments are separated, since an arc in the transitive closure corresponds to a path in the local arc set.

Input: Two segments I and J in the same face f

Output: True if I and J are separated, false otherwise

1: **if** $(r_I, l_J) \in A^*(f)$ **then return** true;
2: **if** $(r_J, l_I) \in A^*(f)$ **then return** true;
3: **if** $(t_I, b_J) \in A^*(f)$ **then return** true;
4: **if** $(t_J, b_I) \in A^*(f)$ **then return** true;
5: **return** false;

We use the above procedure to check whether a pair of shape graphs is complete: Algorithm 4.4 returns true if and only if all pairs of opposite segments in each face of the corresponding orthogonal representation are separated.

Algorithm 4.4 *Completeness test*

Input: Pair of placement graphs (D_x, D_y) for orthogonal representation H with face set F

Output: True if the pair is complete, false otherwise

1: **for all** faces f in F **do**
2: compute $A^*(f)$;
3: **for all** pairs $I \in \mathcal{S}_h(f), J \in \mathcal{S}_v(f)$ **do**
4: **if** not *separated*(I, J) **return** false;
5: **return** true;

Lemma 4.13. *Let H be an orthogonal representation with n vertices. Algorithm 4.4 tests H for completeness in time $O(n^2)$.*

Proof. The construction of the shape graphs $\sigma(H)$ takes linear time. For each face f, the transitive closure of the arc sets can be built in time $O(|V(f)|^2)$, see (Mehlhorn, 1984). Note that the shape graphs are planar and thus the cardinality of the transitive reduction—that influences the running time of the transitive closure computation—is linear in the number of vertices on the boundary of f. The running time of lines 3 and 4 is $O(|\mathcal{S}_h(f)| \cdot |\mathcal{S}_v(f)|)$ since the separation test takes constant time for each segment pair. The total running time of the algorithm is

$$O\Big(\sum_{f \in F} \big(|V(f)|^2 + |\mathcal{S}_h(f)| \cdot |\mathcal{S}_v(f)|\big)\Big) =$$
$$O\Big(\Big(\sum_{f \in F} |V(f)| + |\mathcal{S}(f)|\Big)^2\Big) = O\big((2n + 2|\mathcal{S}|)^2\big) = O(n^2) \ ,$$

which completes the proof of Lemma 4.13. ∎

Our algorithm for maximal unique completion (Algorithm 4.5) is an extension of Algorithm 4.4. For each pair of segments I and J we do not only check for completeness but construct a list $A_{\text{sep}}(I, J)$ that contains the maximally four arcs in the transitive closure that would separate I and J. If this set contains just one arc, we add it to the arc sets of the extension—it is the only possibility to separate I and J and must certainly be part of any complete extension. As discussed above, we have to update the transitive closure after adding the arcs; we repeat the process until each set $A_{\text{sep}}(I, J)$ is either empty or contains at least two arcs.

For an arc (I, J), we define $r(a)$ as the arc (J, I). Line 10 in Algorithm 4.5 checks whether an additional arc (I, J) induces a cycle. A cycle emerges if the path $J \xrightarrow{*} I$ is present, *i.e.*, if the transitive closure of the arc set contains $r(I, J)$.

Lemma 4.14. *Let H be an orthogonal representation with n vertices. Algorithm 4.5 maximally completes the pair of shape graphs $\sigma(H)$. It runs in time $O(n^3)$ in general and in time $O(n^2)$ in case $\sigma(H)$ is complete. After termination, the lists $A_{sep}(I, J)$ contain the pairs of segments whose separation results in a complete extension of $\sigma(H)$. The lists are empty if and only if H is uniquely completable.*

Proof. The running time of the inner loop (lines 3-15) is quadratic in the number of vertices that bound face f since it is an extension of the completeness test and dominated by the time to compute the transitive closure. In the worst case, only one of the lists $A_{\text{sep}}(I, J)$ has cardinality one. Since uniquely completable extensions are subgraphs of some visibility graphs, and thus planar (Lemma 4.7), this may happen at maximum $O(|V(f)|)$ times. Summing up the running times for all faces in a similar manner as in the proof of Lemma 4.13 results in an overall cubic running time.

It is easy to see that the lists A_{sep} are empty if and only if all pairs of segments are separated at the end of the computation. Otherwise, a list $A_{\text{sep}}(I, J)$ contains those arcs that are able to separate I and J (line 11) and cannot be further reduced since no other uniquely separable segment pair exists. ∎

We can use Algorithm 4.5 to devise an exact polynomial-time algorithm for the compaction problems in the special case of uniquely completable orthogonal representations.

Algorithm 4.5 *Maximum unique placement graph completion*

Input: Pair of placement graphs (D_x, D_y) for orthogonal representation H with face set F

Output: Maximally uniquely completed placement graphs $(\widehat{D}_x, \widehat{D}_y)$

```
1:  Â_x = A_x; Â_y = A_y;
2:  for all faces f in F do
3:    repeat
4:      compute Â*(f);
5:      finished = true;
6:      for all pairs I ∈ S_h(f), J ∈ S_v(f) do
7:        if not separated(I, J) then
8:          L_sep = {(r_J, l_I), (r_I, l_J), (t_J, b_I), (t_I, b_J)};
9:          A_sep(I, J) = ∅;
10:         for all a ∈ L_sep do
11:           if r(a) ∉ Â*(f) then A_sep(I, J) = A_sep(I, J) ∪ {a};
12:     for all A_sep(I, J) with |A_sep(I, J)| ≡ 1 do
13:       add a ∈ A_sep(I, J) to Â_x or Â_y; L(I, J) = ∅;
14:       finished = false;
15:   until finished;
```

Theorem 4.5. *Let H be an orthogonal representation with n vertices. If H is uniquely completable, an orthogonal drawing Γ of minimum width $W(\Gamma)$, height $H(\Gamma)$ and area $A(\Gamma)$ or of minimum total edge length $L(\Gamma)$ can be computed in time $O(n^3)$.*

Proof. The proof follows from Theorem 4.1 and its Corollaries 4.1 and 4.1. ∎

4.3.2 Integer Linear Programming Formulation

The polynomial-time algorithm in the previous section computes optimal drawings for instances of the compaction problem in which the corresponding shape graphs are uniquely completable. In general, this is not the case. This section presents integer linear programming (ILP) formulations for the general case—they are the fundament of algorithms for solving the problems that ask for minimum total or minimum maximum edge length to optimality; we also derive formulations for the less important problems of minimizing width, height and half-perimeter. Since the optimization goal in the area minimization variant is the product of width and height and thus non-linear, we cannot provide a direct ILP formulation for this problem. However, optimizing either width or height or half-perimeter of a drawing leads to good area bounds.

Our ILP formulations are based on the following idea: Even in the general case, in which a pair of shape graphs (D_x, D_y) is neither complete nor uniquely completable, Algorithm 4.5 is very useful: At least some pairs of segments may be uniquely separated and in the cases in which the separation is not unique, the test for acyclicity may rule out one or two impossible completions. In the end, the lists $A_{\text{sep}}(I, J)$ keep for each segment pair

of opposite direction two to four arcs one of which must be part of a complete extension of (D_x, D_y). Corollary 4.3 and Lemma 4.4 state that completeness follows by the separation of all segment pairs of opposite direction within each face. The task is now to choose among the arcs in the lists $A_{\text{sep}}(I, J)$ for all such pairs I, J so that two conditions are satisfied:

- The resulting extension must be complete, *i.e.*, an algorithm must choose at least one arc from each set. Nevertheless, the choice of the arcs is non-trivial: The resulting arc set must not introduce a cycle.
- The extension must admit a coordinate assignment which leads to an optimal drawing—in terms of total or maximum edge length, width, height or half-perimeter.

These requirements further restrict the combinatorial formulation of the compaction problems. For a fixed extension of the shape graphs, computing coordinate assignments, which lead to drawings of minimum width, height, total or maximum edge length, reduces to solving linear programs as shown in Chapter 3. In the first three cases, an algorithm must compute minimal or Δ-minimal assignments and the linear programs correspond to shortest path problems or duals of minimum cost flow problems. The fact that LP formulations exist for the fixed case makes it easy to integrate the optimization of the respective aesthetic criterion into a general ILP formulation.

The Compaction Polytope

We will now specify our integer linear programming formulations for the two-dimensional compaction problems. First, we describe how we model the space of extensions, *i.e.*, the set of feasible solutions. The resulting inequalities are part of all following ILP formulations for two-dimensional compaction problems. In a second step, we show how to integrate the different optimization goals, resulting in different formulations. We focus on total edge length minimization, but mention for the other optimization goals where the formulations and the corresponding proofs differ.

To model the space of extensions, we introduce binary variables to decide whether or not the additional arcs are present. We consider only additional arcs which are contained in the lists $A_{\text{sep}}(I, J)$. Since each arc in these lists may or may not be present in a complete extension of the shape graphs, we call these arcs *potential arcs* and refer to them by A_{pot}, *i.e.*,

$$A_{\text{pot}} = \bigcup_{I,J \in N_h \cup N_v} A_{\text{sep}}(I, J) \ .$$

Let H be an instance for the compaction problems and let (D_x, D_y) be the corresponding shape graphs $\sigma(H)$. By Theorem 4.4 we can write the set of feasible solutions of the combinatorial formulation of the compaction problems as

$$\mathcal{C}(D_x, D_y) = \{(\widehat{D}_x, \widehat{D}_y) \mid (\widehat{D}_x, \widehat{D}_y) \text{ complete extension of } (D_x, D_y)\} \ .$$

Let $\mathbb{Q}^{A_{\text{pot}}}$ be the vector space whose elements are indexed with numbers corresponding to the members of A_{pot}. Each complete extension $(\widehat{D}_x, \widehat{D}_y)$ defines an element $x^{(\widehat{D}_x, \widehat{D}_y)} \in$

$\mathbb{Q}^{A_{\text{pot}}}$ of this vector space in the following manner:

$$x_a^{(\widehat{D}_x,\widehat{D}_y)} = \begin{cases} 1 & a \in \widehat{A}_x \cup \widehat{A}_y \\ 0 & \text{otherwise} \end{cases}$$

We use these incidence vectors to characterize the *compaction polytope*

$$P_{\text{COMP}} = \text{conv}\{x^{(\widehat{D}_x,\widehat{D}_y)} \in \mathbb{Q}^{A_{\text{pot}}} \mid (\widehat{D}_x, \widehat{D}_y) \in \mathcal{C}(D_x, D_y)\} \ .$$

In order to model the different coordinate assignments (c_x, c_y) we introduce an additional variable c in the vector space $\mathbb{Q}^S$. Again, the vector space is indexed with numbers corresponding to horizontal or vertical segments with the following interpretation:

$$c_S = \begin{cases} c_x(S) & S \in S_v \\ c_y(S) & S \in S_h \ . \end{cases}$$

We call c the *coordinate vector*.

In our formulations, we describe the polytope P_{COMP} by the following classes of inequalities. Note that we do not require integrality of the coordinate vector; we will show in Observation 4.2 that this is not necessary in order to obtain a grid drawing.

– **Trivial inequalities**. Since x is a characteristic vector, we have the following trivial bounds:

$$0 \leq x_a \leq 1 \qquad \forall a \in A_{\text{pot}} \tag{4.5}$$

– **Integrality constraints**. Additionally, we require x to be integral:

$$x_a \in \{0, 1\} \qquad \forall a \in A_{\text{pot}} \tag{4.6}$$

– **Separation inequalities**. This class of inequalities guarantees completeness of the extension. For each non-separated segment pair of opposite direction within a common face at least one of the paths must be contained in the arc sets, *i.e.*, the sum of the corresponding entries in the characteristic vector must be at least one.

$$\sum_{a \in A_{\text{sep}}(I,J)} x_a \geq 1 \qquad \forall A_{\text{sep}}(I, J) \neq \varnothing \tag{4.7}$$

– **Shape inequalities**. The resulting assignment must be feasible for the extension. For the arcs that are already contained in the shape graphs we have the same inequalities as in the linear programs in Chapter 3:

$$c_J - c_I \geq 1 \qquad \forall (I, J) \in A_x \cup A_y \tag{4.8}$$

– **Consistency inequalities**. If a potential arc $a = (I, J)$ is part of the extension, then the corresponding constraint must be respected. In this case, the inequality must guarantee the feasibility of the assignment, *i.e.*, it must be a shape inequality. If the arc does not belong to the extension, the corresponding constraint must not restrict the set of feasible solutions. With each potential arc $a \in A_{\text{pot}}$ we associate a large number M_a which enables us to switch between trivially satisfied inequalities and shape inequalities.[10]

$$c_J - c_I - (M_a + 1)x_a \geq -M_a \qquad \forall a = (I, J) \in A_{\text{pot}} \tag{4.9}$$

First, let $x_a = 1$: Then inequality (4.9) changes to

$$c_J - c_I \geq -M_a + M_a + 1 = 1 \tag{4.9a}$$

and is of type (4.8). In case $x_a = 0$, we get

$$c_J - c_I \geq -M_a \ . \tag{4.9b}$$

This inequality is satisfied if M_a is large enough. In other words, M_a must be an upper bound on the distance $c_I - c_J$. The following lemma defines the choice of the constant M_a.

Lemma 4.15. *The value max$\{|S_h|, |S_v|\}$ is an upper bound on the distance $c_I - c_J$ and thus a sufficient choice for the constants M_a in the consistency inequalities.*

Proof. Note that any optimally compacted drawing Γ has width $W(\Gamma) \leq |S_v|$ and height $H(\Gamma) \leq |S_h|$, otherwise a one-dimensional compaction step could be applied. The distance $c_I - c_J$ corresponds to the distance of either two x- or two y-coordinates in Γ. Clearly, no such distance can be greater than the maximum of width and height. ∎

Having introduced the different classes of inequalities, we present the ILP formulations for the combinatorial equivalents of the two-dimensional compaction problems. We begin with the formulation for L-COMP, which corresponds to minimizing the total edge length, and prove that an optimal solution of the ILP indeed results in an optimal drawing of the original orthogonal representation.

Minimizing Total Edge Length

Let H be an instance of L-COMP and let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be the shape graphs $\sigma(H)$. We construct the following integer linear programming formulation for L-COMP. The set of inequalities is exactly as described above, we optimize over the space of complete extensions of the shape graphs. The objective function requires that both assignments are Δ-minimal, notice the similarity to the linear program (LP.3) for the fixed case on page 34.

[10] In the literature on ILP modeling, this approach is also known as the "big M approach".

$$\min \quad \sum_{(I,J)\in A_x\cup A_y} c_J - c_I \tag{ILP.1}$$

$$\text{subject to} \quad \sum_{a\in A_{\text{sep}}(I,J)} x_a \geq 1 \qquad \forall A_{\text{sep}}(I,J) \neq \varnothing \tag{ILP.1.1}$$

$$c_J - c_I \geq 1 \qquad \forall (I,J) \in A_x \cup A_y \tag{ILP.1.2}$$

$$c_J - c_I - (M_a + 1)x_a \geq -M_a \qquad \forall a = (I,J) \in A_{\text{pot}} \tag{ILP.1.3}$$

$$0 \leq x_a \leq 1 \qquad \forall a \in A_{\text{pot}} \tag{ILP.1.4}$$

$$x_a \in \{0,1\} \qquad \forall a \in A_{\text{pot}} \tag{ILP.1.5}$$

The following observation shows that we do not need to require integrality of the coordinate vector c. For a fixed characteristic vector x, the problem is identical to two static problems which can be solved by the algorithms in Chapter 3.

Observation 4.2. *Let (x, c_f) with $x \in \mathbb{Q}^{A_{pot}}$ and $c_f \in \mathbb{Q}^S$ be a feasible solution of (ILP.1) and let z_f be the value of the objective function. Then there is also a feasible solution (x, c) with $c \in \mathbb{Z}^{S_h \cup S_v}$ and objective function value $z \leq z_f$.*

Proof. Since x is part of a feasible solution, its components must be either zero or one. Then the integer linear program reduces to computing two separate Δ-minimal assignments; the consistency constraints can be either eliminated (if the corresponding variable $x_a = 0$) or turn to shape constraints (if $x_a = 1$). The result follows with Theorem 3.5. ∎

In our characterization of integral points within the compaction polytope P_{COMP} by inequalities (4.5)-(4.9) we do not explicitly exclude cycles in the extension. The following lemma shows that the consistency inequalities suffice to ensure the absence of cycles in the solution, so no additional inequalities are necessary. In the description of the branch-and-cut algorithm in Section 4.3.3 we will see, however, that adding a class of *cycle inequalities* results in a tighter description of the compaction polytope P_{COMP}.

Lemma 4.16. *Let (x, c) be a feasible solution of (ILP.1) and let $\widehat{D}_x$ and $\widehat{D}_y$ be the extension corresponding to x. Then $\widehat{D}_v$ and $\widehat{D}_y$ are acyclic.*

Proof. Assume that there is a cycle $C = \big((S_1,S_2),(S_2,S_3),\ldots,(S_{k-1},S_k),(S_k,S_1)\big)$ of length k in the extension. Note that each arc $a = (I,J)$ in C belongs either to $A_x \cup A_y$ or to A_{pot}. Observe that if $a \in A_x \cup A_y$, then the ILP contains an appropriate constraint $c_J - c_I \geq 1$. For the potential arcs on C, we must have $x_a = 1$ for all $a = (I,J) \in C \cap A_{\text{pot}}$, and the corresponding consistency constraint turns to

$$\begin{aligned} & c_J - c_I - (M_a + 1) \geq -M_a \\ \Leftrightarrow \qquad & c_J - c_I \geq 1 \ . \end{aligned}$$

For the arcs on the cycle, this means

$$\begin{aligned} c_{S_2} - c_{S_1} &\geq 1 \\ c_{S_3} - c_{S_2} &\geq 1 \\ &\vdots \\ c_{S_1} - c_{S_k} &\geq 1 \end{aligned}$$

The left sides of the inequalities sum up to zero, the right side sums up to $k > 0$ which results in the contradiction $0 > k > 0$. ∎

To construct the integer linear program, we start with the input of the compaction problem L-COMP, an orthogonal representation H. We construct the shape graphs $\sigma(H)$ and determine the maximally unique completion of $\sigma(H)$. This process results in a set of potential arcs in the lists $A(I, J)$ for which we can build the above ILP. We have to find a complete extension of the shape graphs for which we can compute the best pair of Δ-minimal assignments—this is the combinatorial equivalent of L-COMP as given in Definition 4.12(a). It remains to show that there is a one-to-one correspondence between feasible solutions of the combinatorial problem and feasible solutions of the integer linear program.

Theorem 4.6. *Let H be an orthogonal representation. A feasible solution (x, c) of (ILP.1) for H corresponds to a feasible solution of the combinatorial version of problem L-COMP and thus to a drawing Γ of H and vice versa. The total edge length $L(\Gamma)$ in the drawing is equal to the value of the objective function.*

Proof. For the first part of the proof, let x and c be the solution vectors of (ILP.1). According to Observation 4.2 we can assume that both vectors are integer. Vector x describes a complete extension $(\widehat{D}_x, \widehat{D}_y)$ of $\sigma(H)$; inequalities (4.7) guarantee the separation of segments, the absence of cycles follows by Lemma 4.16, hence the extension is complete. The consistency inequalities (4.9) require c to correspond to a coordinate assignment for $(\widehat{D}_x, \widehat{D}_y)$. The result follows with Theorems 4.1 and 4.4.

To prove the other direction we construct a solution of the combinatorial problem version by choosing the arcs according to the visibility graphs $\nu(\Gamma)$ and setting the binary variables x accordingly. Likewise, we set the entries of the coordinate vector c to the actual coordinates in Γ. It is easy to verify that the pair (x, c) does not violate any of the constraints in (ILP.1).

Each arc in the arc sets of the shape graphs $A_x \cup A_y$ corresponds to a horizontal or vertical edge in the orthogonal representation H. Since c corresponds to a coordinate assignment for the extension, it also corresponds to a coordinate assignment for the shape graphs $\sigma(H)$. The distance of an arc in the constraint graphs is equal to the length of the appropriate edge. Hence, the value of the objective function is equal to the sum of the edge lengths. ∎

Corollary 4.4. *An optimal solution (x^*, c^*) of (ILP.1) corresponds to an optimal drawing for H with respect to total edge length.*

Minimizing Width, Height, and Half-Perimeter

In order to compute drawings of minimal width or height, we have to find a complete extension of the given shape graphs $\sigma(H)$ and a coordinate vector c which corresponds to a minimal assignment for the respective constraint graph. We use the same technique as in the static case in Section 3.1 and add a super source s and a super sink t.

Let s_x, t_x and s_y, t_y be the additional nodes in the horizontal and vertical shape graphs D_x and D_y, respectively. Like in the static case, we add the arc sets

$$A_{s_x} = \{(s_x, I) \mid I \in N_x\} \qquad A_{t_x} = \{(I, t_x) \mid I \in N_x\}$$
$$A_{s_y} = \{(s_y, I) \mid I \in N_y\} \qquad A_{t_y} = \{(I, t_y) \mid I \in N_y\}$$

and set the weight $w(a)$ of each $a \in A_{s_x} \cup A_{t_x} \cup A_{s_y} \cup A_{t_y}$ to zero.

We get the following objective functions $z : c \to \mathbb{Q}$, depending on the optimization goal:

$$z(c) = \begin{cases} c_{t_x} - c_{s_x} & \text{when minimizing the width } W(\Gamma) \\ c_{t_y} - c_{s_y} & \text{when minimizing the height } H(\Gamma) \\ c_{t_x} - c_{s_x} + c_{t_y} - c_{s_y} & \text{when minimizing the half-perimeter } P(\Gamma) \end{cases}$$

The integer linear programs look as follows:

$$\begin{aligned}
\min \quad & z(c) && && \text{(ILP.2)} \\
\text{subject to} \quad & \sum_{a \in A_{\text{sep}}(I,J)} x_a \geq 1 && \forall A_{\text{sep}}(I,J) \neq \varnothing && \text{(ILP.2.1)} \\
& c_J - c_I \geq 1 && \forall a = (I,J) \in A_x \cup A_y && \text{(ILP.2.2a)} \\
& c_J - c_I \geq 0 && \forall a = (I,J) \in A_{s_x} \cup A_{t_x} && \text{(ILP.2.2b)} \\
& c_J - c_I \geq 0 && \forall a = (I,J) \in A_{s_y} \cup A_{t_y} && \text{(ILP.2.2c)} \\
& c_J - c_I - (M_a + 1)x_a \geq -M_a && \forall a = (I,J) \in A_{\text{pot}} && \text{(ILP.2.3)} \\
& 0 \leq x_a \leq 1 && \forall a \in A_{\text{pot}} && \text{(ILP.2.4)} \\
& x_a \in \{0,1\} && \forall a \in A_{\text{pot}} && \text{(ILP.2.5)}
\end{aligned}$$

Of course, we can omit the additional arcs in one of the constraint graphs and the corresponding shape inequalities if the aim is a drawing of minimal width or height. A similar theorem as for (ILP.1) states that solutions of the above integer linear programs correspond to feasible drawings.

Theorem 4.7. *Let H be an orthogonal representation. A feasible solution (x, c) of (ILP.2) for H corresponds to a feasible solution of the combinatorial versions of problems W-COMP, H-COMP and P-COMP and thus to a drawing Γ of H and vice versa. The width $W(\Gamma)$, height $H(\Gamma)$ or half-perimeter $P(\Gamma)$ of the drawing are equal to the value of the objective function z.*

Proof. The proof is analogous to the proof of Theorem 4.6. ∎

Corollary 4.5. *An optimal solution (x^*, c^*) of (ILP.2) corresponds to an optimal drawing for H with respect to width, height or half-perimeter, depending on the choice of the objective function.*

Minimizing Maximum Edge Length

As for the other criteria, the integer linear program whose optimal solutions correspond to drawings with minimum maximum edge length is inspired by the static case. In Section 3.3 we show how to compute minimax assignments by introducing a new variable whose value is an upper bound for each arc distance. We do the same here and add $l_{\max} \in \mathbb{Q}$ to the set of variables. Like in Section 3.3, we have an additional set of inequalities that forces the value of $l_{\max}$ to be at least as big as the longest horizontal and vertical edge. Minimizing $l_{\max}$ then corresponds to minimizing the length of the longest edge in a drawing Γ for H.

$$\begin{aligned}
\min \quad & l_{\max} && && \text{(ILP.3)}\\
\text{subject to} \quad & \sum_{a \in A_{\text{sep}}(I,J)} x_a \geq 1 && \forall A_{\text{sep}}(I,J) \neq \varnothing && \text{(ILP.3.1)}\\
& c_J - c_I \geq 1 && \forall (I,J) \in A_x \cup A_y && \text{(ILP.3.2)}\\
& c_J - c_I - l_{\max} \leq 0 && \forall (I,J) \in A_x \cup A_y && \text{(ILP.3.3)}\\
& c_J - c_I - (M_a + 1)x_a \geq -M_a && \forall a = (I,J) \in A_{\text{pot}} && \text{(ILP.3.4)}\\
& 0 \leq x_a \leq 1 && \forall a \in A_{\text{pot}} && \text{(ILP.3.5)}\\
& x_a \in \{0,1\} && \forall a \in A_{\text{pot}} && \text{(ILP.3.6)}
\end{aligned}$$

Theorem 4.8. *Let H be an orthogonal representation. A feasible solution (x, c) of (ILP.3) for H corresponds to a feasible solution of the combinatorial version of problem l_{max}-COMP and thus to a drawing Γ of H and vice versa. The maximum edge length $l_{max}(\Gamma)$ is equal to the value of the objective function.*

Proof. The proof is analogous to the proof of Theorem 4.6. ∎

Corollary 4.6. *An optimal solution (x^*, c^*) of (ILP.3) corresponds to an optimal drawing for H with respect to maximum edge length.*

We conclude the section on integer linear programming formulations for the two-dimensional compaction problems by two remarks on minimizing the longest edge.

Remark 4.7. An optimal solution of (ILP.3) corresponds to a drawing with minimum maximum edge length among all drawings for the given simple orthogonal representation H. However, the total edge length in this drawing may be quite high. For practical purposes, it makes sense to optimize the total edge length as a secondary goal. We model this by changing the objective function according to the new optimization goal. Inequalities (ILP.3.3) ensure that each edge is shorter than the maximum edge length, this implies that the total length of edges in E is lower than, or equal to $l_{\max}$ times the number of edges $|E|$. The objective function is now

$$\min |E| \cdot l_{\max} + \sum_{(I,J) \in A_x \cup A_y} c_J - c_I \ .$$

Remark 4.8. Our formulation in (ILP.3) is correct if the input is truly a simple representation. In general, however, the input represents a non-simple representation with artificial vertices instead of the original bends. In this case, (ILP.3) only minimizes the length of the longest edge segment and not of the longest edge in the drawing. In the following, we present a generalization of (ILP.3) which also works for non-simple representations.

The two-dimensional nature of our problem formulation enables us to consider all segments of a bent edge: Let e be an edge in H and let $A(e)$ be the arcs in the shape graphs which correspond to the edge segments of e. Changing (ILP.3.3) to

$$\sum_{(I,J)\in A(e)} (c_J - c_I) - l_{\max} \leq 0 \qquad \forall e \in H \tag{ILP.3.3b}$$

takes into account that an edge may be composed of several edge segments in the original graph.

4.3.3 Branch-and-Bound and Branch-and-Cut Algorithm

The integer linear programs in the preceding section enable us to use the generic branch-and-bound and branch-and-cut algorithms of Chapter 2 to find optimal solutions of the two-dimensional compaction problems.

Common to both algorithms is the following preprocessing phase: Let H be an instance of a compaction problem. In a first step, we determine the corresponding shape graphs $\sigma(H)$ and compute their maximal unique completion with Algorithm 4.5. This phase has two objectives:

1. It detects if the shape graphs are complete or uniquely completable. In this case, we can solve the compaction problems in polynomial time by Theorem 4.5.

2. In the general case, Algorithm 4.5 makes the compaction problems easier since it separates segments for which the relative placement is determined in advance. Furthermore, it constructs the sets A_{sep} for the unseparated segments. After execution, the lists $A_{\text{sep}}(I, J)$ contain for each unseparated pair of segments I and J those arcs which may be added in order to separate them. The maximally completed shape graphs together with the potential arcs in the sets A_{sep} determine the integer linear program for the respective compaction problem.

We will first describe the linear programming-based branch-and-bound algorithm. At each node of the branch-and-bound tree we compute a local lower bound and try to improve the global upper bound (see the generic algorithm on page 26 in Chapter 2 and note that the compaction problems are minimization problems).

In the initialization phase we try to compute a good initial global upper bound. We apply one of the constructive heuristics in Section 4.2.4 and improve the first feasible solution with an improvement heuristics described in Section 4.2.5. According to the generic scheme we initialize the list of subproblems with the original problem.

At each node of the branch-and-bound tree we obtain local lower bounds by solving the LP-relaxations of the appropriate subproblem, *i.e.*, the linear programs resulting from

dropping the integrality constraints. If the objective value of the LP exceeds the global upper bound, we know that the feasible solutions contained in the subtree rooted at the current node are worse than the currently best solution, and we fathom the subproblem. If this is not the case, we may have found a better solution: If the LP-solution is feasible for the original problem, we can update the global upper bound. Otherwise we add two new subproblems to the list by choosing a zero-one variable and fixing it to zero in the first subproblem and to one in the second. The branching step corresponds to the decision of adding or not adding a certain potential arc to the extension. By Observation 4.2 we do not have to branch on the coordinate variables, since their integrality follows from the integrality of the potential arc variables.

The algorithm then selects an open problem from the list of subproblems according to a certain enumeration strategy, like, *e.g.*, depth-first-search, breadth-first-search or best-first-search. We comment on our choice for the branching and selection strategy in the description of the realization of our algorithm in Section 4.4.1. If the list of subproblems is empty, the global upper bound is an optimal solution.

Cutting Planes

We extend the integer linear programming formulation by adding an additional class of constraints that do not eliminate feasible solutions and improve the bound of the LP relaxation by cutting off some of the fractional solutions.

Although Lemma 4.16 shows that explicitly excluding cycles is not necessary in the formulation, we will show that the *cycle inequalities* are tighter than the corresponding sets of consistency inequalities.

– **Cycle inequalities**. All cycles in extensions of shape graphs have positive weight since all arcs have unit weight. We know that the initial shape graphs are upward planar and thus acyclic. Therefore, we explicitly exclude cycles that are induced by the additional potential arcs. For each cycle C in the extension we require that not all of the potential arcs on C may be present at once.

$$\sum_{C \cap A_{\text{pot}}} x_a \leq |C \cap A_{\text{pot}}| - 1 \qquad \forall \text{cycles } C \text{ in extension} \tag{4.10}$$

The disadvantage of the cycle inequalities is their possibly exponential number that prevents us from adding them in advance to the ILP formulation. Instead, we separate violated inequalities of this class at each node of the branch-and-bound tree which results in a branch-and-cut algorithm.

We first show that the new class of inequalities cuts off fractional infeasible solutions that are valid for the consistency constraints and then investigate the corresponding separation problem.

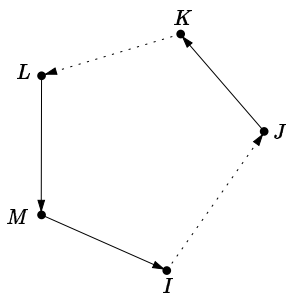
We consider the relation of cycle inequalities and consistency inequalities using a global constant M with $M_a = M$ for all $a \in A_{\text{pot}}$ as in our choice in Lemma 4.15

on page 83. Summing up the consistency inequalities of a cycle results in

$$\begin{aligned} -(M+1)\sum_{a\in C\cap A_{\text{pot}}} x_a &\geq |C\cap A_{\text{pot}}|\cdot(-M)+|C|-|C\cap A_{\text{pot}}| \\ \Leftrightarrow \qquad \sum_{a\in C\cap A_{\text{pot}}} x_a &\leq \frac{M\cdot|C\cap A_{\text{pot}}|+|C\cap A_{\text{pot}}|-|C|}{M+1} \\ &= \frac{(M+1)|C\cap A_{\text{pot}}|-|C|}{M+1} \\ &= |C\cap A_{\text{pot}}|-\underbrace{\frac{|C|}{M+1}}_{<1} > |C\cap A_{\text{pot}}|-1 \ , \end{aligned}$$

a weaker inequality than the corresponding cycle inequality.

This calculation also shows that the constants M_a determine the difference in the strength of the inequalities. A good bound for M_a will weaken the disadvantage of the consistency constraints. Their advantage is, however, that we only need $|A_{\text{pot}}|$ constraints as compared to the possibly exponential number of cycle constraints. Figure 4.23 shows the relation between the classes of inequalities with an example.



$M_{(I,J)} = M_{(K,L)} = 9$, $w(a) = 1$ for all arcs

consistency inequalities:

$$\begin{aligned} c_J - c_I - 10x_{(I,J)} &\geq -9 && (4.11)\\ c_K - c_J &\geq 1 && (4.12)\\ c_L - c_K - 10x_{(K,L)} &\geq -9 && (4.13)\\ c_M - c_L &\geq 1 && (4.14)\\ c_I - c_M &\geq 1 && (4.15) \end{aligned}$$

Summing up (4.11)-(4.15) results in

$$\begin{aligned} -10(x_{(I,J)}+x_{(K,L)}) &\geq -15 \\ \Leftrightarrow \qquad x_{(I,J)}+x_{(K,L)} &\leq 1.5 \end{aligned}$$

cycle inequality:

$$x_{(I,J)}+x_{(K,L)} \leq 1$$

Figure 4.23: Example showing that cycle inequalities are tighter than consistency inequalities. Potential arcs are dashed

Cycle inequalities play a more important role in the next chapter on labeling in which we develop a pure zero-one formulation for point-feature map labeling problems. Unfortunately, the separation problem becomes difficult in our formulations for the labeling problems since not every cycle has positive weight. We will show in Section 5.3.3 that it is *NP*-hard to separate violated positive cycle inequalities in the case of general arc weights.

This task is easier in our formulations for the compaction problems. For the sake of simplicity, we assume that additional zero-one variables are associated with each arc in the shape graphs and that these variables are fixed to one during the computation. Since every cycle in the extension has positive weight, the separation problem is equivalent to the following problem.

Definition 4.14 (Separation problem for cycle inequalities).
Given a digraph $D = (N, A)$ and an LP-solution $\chi \in \mathbb{Q}^A$, $0 \leq \chi_a \leq 1 \ \forall a \in A$, find a directed cycle $C \subseteq A$ with

$$\sum_{a \in C} \chi_a \ > \ |C| - 1 \ .$$

We substitute $\xi_a = 1 - \chi_a$ for all $a \in A$. We then have to find a directed cycle with

$$\begin{aligned} & \sum_{a \in C} (1 - \xi_a) \ > \ |C| - 1 \\ \Leftrightarrow \qquad & \sum_{a \in C} \xi_a \ < \ 1 \ . \end{aligned}$$

We propose the following, straight-forward shortest path-based technique to solve the separation problem: Let the length of an arc be its ξ-value. We temporarily remove each arc $a = (i, j)$ from the graph and check whether the length $\xi(P)$ of the shortest path P from j to i plus the length of the arc a is smaller than one. In this case, the cycle formed by $P \cup \{a\}$ gives rise to a violated cycle inequality. Of course, we only have to consider the potential arcs since no cycles without potential arcs exist. Furthermore, we can use Dijkstra's algorithm to compute shortest paths since all arc weights are nonnegative. Algorithm 4.6 summarizes the steps of the cycle separation.

Algorithm 4.6 *Cycle separation*

Input: Directed graph $D = (N, A)$, LP-solution $\chi \in \mathbb{Q}^A$, $0 \leq \chi_a \leq 1 \ \forall a \in A$
Output: Directed cycle $C \subseteq A$ with $\sum_{a \in C} \chi_a > |C| - 1$ if one exists

```
1: for all a = (i, j) ∈ A do
2:    A = A \ {a};
3:    compute shortest path P from j to i with Dijkstra;
4:    if ξ(P) + ξ(a) < 1 then
5:       return P ∪ {a};
6:    A = A ∪ {a};
```

Although we can separate violated cycle inequalities quite efficiently by using Algorithm 4.6, we still have to keep the consistency constraints in order to link the coordinates to the potential arcs. Practical experiments with the above branch-and-cut algorithm show that the number of violated inequalities that can be found decreases quickly as the computation proceeds. After a few iterations in the branch-and-cut scheme no further violating

cycles exist, but the algorithm still spends a considerable part of the computation time in the separation procedure. At each node in the tree this amounts to an unsuccessful shortest path computation for every arc in A_{pot}.

We conclude that the pure branch-and-bound algorithm is more suited to solve the ILP formulations presented in this chapter. We believe that a more promising way to increase the performance of the algorithm consists of finding better bounds for the differences $c_I - c_J$ for each potential arc $a = (I, J)$. This would enable us to compute individual constants M_a that improve the tightness of the consistency constraints. We want to remark, however, that the implementation of the branch-and-bound algorithm with the choice of a global constant M according to Lemma 4.15 is sufficiently fast for practical instances. Our experimental study in Section 4.4 shows that all practical problem instances can be solved in short computation time; the longest running time for a practical instance is ten seconds.

4.3.4 Related Work in VLSI Design

The algorithms by Kedem and Watanabe (1984) and by Schlag *et al.* (1983) are exact algorithms for a two-dimensional compaction problem in VLSI design which is closely related to A-COMP. In both cases, the authors propose a branch-and-bound approach to solve the problems.

Kedem and Watanabe (1984) propose a translation of the two-dimensional compaction problem into a nonlinear mixed integer programming formulation, Watanabe (1984) provides a detailed description of the algorithm. The authors express the problem as minimizing the nonlinear area function under a set of linear and nonlinear constraints. Their formulation, however, sacrifices the general statement of A-COMP as defined in Section 4.1 and considers only a subset of the feasible solutions in order to achieve a better running time. In the following we sketch the branch-and-bound algorithm.

A vector of decision variables d determines the interaction of components. In the given formulation, only two positions are possible for a pair of components, coded as an entry in a zero-one-vector d. Each combination corresponds to a different relative placement of the components in this pair—either a horizontal or a vertical constraint is active. An entry in d can be interpreted as two potential arcs in the constraint graphs, *i.e.*, a fixed d specifies two (possibly infeasible) one-dimensional compaction problems. This formulation has the drawback of being able to handle only two-way choices instead of four possible relative placements. Though the area of the computed layout is optimal for a given partial order, it may not be optimal for an instance of A-COMP as formulated in Section 4.1. The authors propose a post-processing step to determine where the partial order of elements has to be swapped, but they do not present a method that guarantees an overall optimal solution.

At each node in the branch-and-bound tree, the two one-dimensional compaction problems are solved by computing a minimal coordinate assignment with the longest-path method given in Algorithm 3.1. If the subproblem is infeasible, *i.e.*, Algorithm 3.1 reports a positive cycle, the tree of problems can be cut at this node. If the node is a leaf and a feasible solution is found that is better than the previous global upper bound, the bound is updated. Otherwise, a solution may cause an update of the local lower bound.

If the latter becomes greater than the global upper bound, an optimal solution cannot be found below the current node, and again the tree is cut at this point.

In general, an optimal solution of the restricted problem can be found in short time by using this method. However, the proof of optimality may be very time-consuming.

Schlag *et al.* (1983) propose a different branch-and-bound approach: They present a characterization of feasible layouts in terms of satisfiability of a special Boolean expression.

If the distance constraints between two components i and j in the layout process are not satisfied, the pair is called a *violation*. Four logical constraints $c_{ij}^1, c_{ij}^2, c_{ij}^3$, and c_{ij}^4 define the relative placement of elements i and j. A violation can be seen as the non-emptiness of the intersection between the two rectangles R_{ij} and j, shown in Figure 4.24. The rectangle R_{ij} contains element i; at each of the four sides it is enlarged by the appropriate minimum distance to element j.

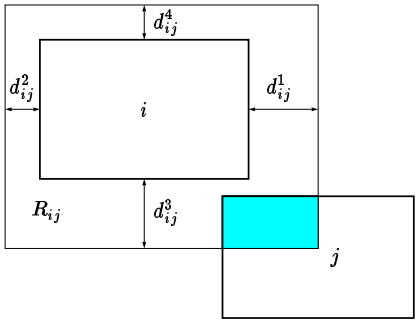


Figure 4.24: A violation formed by elements i and j

With each logical constraint

$$c \in \bigcup_{1 \le i < j \le n} \{c_{ij}^1, c_{ij}^2, c_{ij}^3, c_{ij}^4\}$$

and each layout P, the authors associate a logical variable. The variable is true if the logical constraint is satisfied in P and false otherwise. Then the following properties characterize a legal layout:

- Layout P satisfies the base constraints which determine the sizes of the elements.
- For every c the formula F is true, where

$$F = \bigwedge_{i,j} \left(c_{ij}^1 \vee c_{ij}^2 \vee c_{ij}^3 \vee c_{ij}^4 \right) \quad .$$

For a practical application, the size of set F is too big. The basic idea of the approach by Schlag *et al.* is to start with $F = \varnothing$, and to obtain a so-called *smashing* by solving the system of inequalities with the longest path method in the corresponding constraint

graphs. A smashing is a possibly illegal layout that respects only the set of constraints in the current set F. Transferred to the area of graph drawing, this corresponds to a drawing obtained from a minimal assignment for the shape graphs of an orthogonal representation, similar to Figure 4.20. Then the algorithm determines a violation (i, j) in the smashing with a rectangle intersection algorithm, searching for situations like in Figure 4.24. Once such a pair (i, j) is found, the algorithm performs a branching step: in each of the four subproblems, it adds a different logical constraint c_{ij}^k ($k \in \{1, \ldots, 4\}$) to the set of active constraints F. It then proceeds recursively with each of the four different sets $F + \{c_{ij}^1\}, \ldots, F + \{c_{ij}^4\}$.

An optimal solution is obtained like in the algorithm by Kedem and Watanabe (1984). If the subproblem is a leaf, the generated logical constraints are either inconsistent, or the layout is legal and may constitute the new upper bound. Computations at inner nodes in the branch-and-bound tree have the following effects: illegal subproblems and problems exceeding the global upper bound cause the algorithm to cut the tree at the current node; otherwise, the area of the smashing becomes the new local lower bound.

Unlike our algorithms, the exact branch-and-bound algorithms from the area of VLSI design aim at area minimization for related, but different, problems and rest upon non-linear formulations and objective functions. The LP-based relaxations, which are the core of our new techniques, cannot be used in a non-linear environment. Furthermore, we take advantage of the particular properties of the compaction problems as they occur in orthogonal graph drawing. We characterize the set of feasible solutions as integral points within a problem-specific polytope and exploit the properties of our combinatorial reformulation to optimize over this polytope. The running time behavior of our algorithms benefits in particular from the small number of *potential arcs* which give rise to binary variables in our formulations. We are able to identify a small set of potential arcs by maximally completing the shape graphs.

4.4 Experimental Study

In this section we report on our experimental results for the two-dimensional compaction problems in orthogonal graph drawing. We describe the implementation of the algorithms under evaluation and specify the experimental settings. We apply two variants of the GIOTTO algorithm to a large number of graphs from three different test suites to create the instances of the compaction problems. We discuss the experimental results with a special focus on edge length and running time—we demonstrate, however, that the other criteria result in a similar behavior. Moreover, we dedicate a paragraph to turn-regularity-related issues. At the end of this section, we summarize the main conclusions from this experimental study.

4.4.1 Implementations

All compaction strategies introduced in the preceding sections are available as modules inside the AGD library (see AGD, 2001). AGD is an object-oriented C++ class library of algorithms and data structures which aims at bridging the gap between theory and practice in

the field of automatic graph drawing. Gutwenger (1999) and Alberts, Gutwenger, Mutzel, and Näher (1997) present the design of the library, and Gutwenger, Jünger, Klau, Leipert, and Mutzel (2000) describe its usage as a tool for algorithm engineering. AGD is built on top of LEDA, a library of efficient data types and algorithms in the area of combinatoric and geometric computing. The book (Mehlhorn and Näher, 1999) contains a comprehensive and detailed description of the library.

The design of algorithms within AGD is very modular. The compaction phase in orthogonal graph drawing is strictly separated from the computation of the orthogonal shape and provides a large number of different strategies: The user can choose among a constructive heuristics and optionally combine it with an improvement heuristics. Alternatively the user can use one of the exact algorithms. The following modules exist in the library:

- **Constructive heuristics.** A different choice of dissection method and assignment leads to different constructive heuristics as described in Section 4.2.4. We provide a variant of the original rectangle-based and the two turn-regularity based dissection methods. We can assign coordinates to the auxiliary representations by either minimal or Δ-minimal assignments which results in six different constructive heuristics. In this section, we test four of them: rectangular dissection with minimal and Δ-minimal assignment, and the two turn-regularity-based dissection methods with Δ-minimal assignments. We refer to the implementations by LP (longest paths), FL (flow), TR1 (turn-regularity 1) and TR2 (turn-regularity 2), respectively.

- **Improvement heuristics.** Two constraint-graph based improvement heuristics are available; they differ in computing either minimal or Δ-minimal assignments for the visibility graphs (see Section 4.2.5). If we apply an improvement heuristics, we append either LP or FL to the name of the constructive heuristics.

- **Exact algorithms.** Our implementations of the exact branch-and-bound algorithms in Section 4.3.3 use CPLEX, a library of C routines that solves (integer) linear programming problems (CPLEX, 1999). An implementation of the branch-and-cut algorithm described in the same section is available which uses the ABACUS-library. ABACUS supports the development of linear-programming based branch-and-bound and branch-and-cut algorithms and provides an interface to CPLEX. For the design of the library see (Jünger and Thienel, 2000) and (Thienel, 1995). We use the pure CPLEX-version for the experiments in this section since it proves to be faster due to the reasons mentioned at the end of Section 4.3.3. We refer to the exact algorithms which minimize total and maximum edge length by L-OPT and $l_{\max}$-OPT, respectively. The objective function in the implementation $l_{\max}$-OPT corresponds to minimizing total edge length as a secondary optimization goal as discussed at the end of Section 4.3.2.

We test the implementations of the constructive heuristics both stand-alone and in combination with implementations of the two constraint graph-based improvement heuristics which results in twelve different heuristics for the two-dimensional compaction problems of Section 4.1. Additionally, we test the implementation of the exact integer linear

programming-based algorithms for L-COMP and $l_{\max}$-COMP. Table 4.2 summarizes the naming scheme for the 14 resulting algorithms under evaluation.

For the exact branch-and-bound algorithms, we set a limit of 15 minutes CPU-time. If the computation exceeds this limit without finding an optimal solution, we return the best solution together with its quality guarantee.

	Improvement heuristics		
Constructive heuristics	none	minimal ass.	Δ-minimal ass.
rectangular dissection, minimal assignment	LP	LPLP	LPFL
rectangular dissection, Δ-minimal assignment	FL	FLLP	FLFL
turn-regular dissection (var. 1), Δ-minimal assignment	TRI	TRILP	TRIFL
turn-regular dissection (var. 2), Δ-minimal assignment	TR2	TR2LP	TR2FL
Exact algorithms			
optimal w.r.t. total edge length		L-OPT	
optimal w.r.t. max. edge length		$l_{\max}$-OPT	

Table 4.2: Naming scheme for implementations

4.4.2 Experimental Settings

For our experiments, we use three different groups of graphs which can roughly be divided in easy instances, practical instances and hard instances. All instances are publically available at `http://www.ads.tuwien.ac.at/compaction`.

The first group of graphs are a set of relatively easy to compact four-planar biconnected graphs. We use a set of 500 graphs with 10 to 100 vertices. Bridgeman *et al.* (2000) exclusively use four-planar biconnected graphs for their experiments.

In the experimental comparison paper (Di Battista, Garg, Liotta, Tamassia, Tassinari, and Vargiu, 1997), the authors introduce 11,582 practical graphs which we use as the second group of test instances. The graphs are variations from a core set of 112 graphs used in real-world applications from software engineering and database design. In experimental graph drawing, these graphs are a widely-used test-suite and appear in many computational studies.

Finally, we want to test the algorithms on hard instances. We produce 565 instances for the compaction problems with a graph generator in LEDA: To create a planar graph with n vertices, the generator chooses n line segments whose endpoints have random coordinates of the form x/k, where k is the smallest power of two with $k \geq n$, and x is a random integer in $\{0, \ldots, k-1\}$. It then constructs the arrangement defined by the segments and keeps the vertices with the n smallest x-coordinates. Finally, the generator adds edges

to make the graph connected while maintaining its planarity. We call this test-suite *quasi-trees* since large subgraphs of the resulting graphs are trees. Quasi-trees are hard instances of the compaction problem due to their large number of fundamentally different drawings that make the compaction task difficult. The suite consists of a set of 75 smaller quasi-trees with 40 to 80 vertices on which all algorithms are tested and a set of 490 graphs with 100 to 2,500 vertices to challenge the heuristic algorithms.

In order to get instances for the two-dimensional compaction problems of Section 4.1 we compute a simple orthogonal representation for each of the graphs in the test-suites following a transformation that is part of many orthogonal drawing techniques used in practice:

1. For each graph G in the test-suites we compute a planarized graph G' using the planarization method in the AGD library.[11] The number of vertices in G' is equal to the number of vertices in G plus the number of crossings. We then compute a planar embedding of G'.

2. We run the transformation phase of the GIOTTO algorithm as described in (Tamassia *et al.*, 1988). The phase creates an auxiliary four-planar graph G'_4 by replacing vertices of degree greater than four by artificial faces.

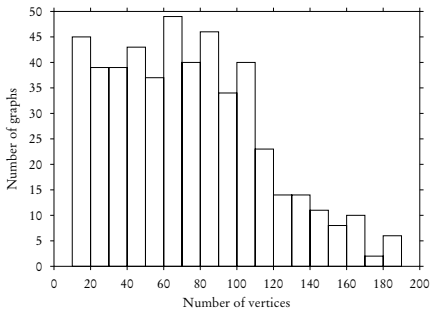
3. We use both the original algorithm by Tamassia (1987) and a variant of this method (Klau, 2001) to create instances of two-dimensional compaction problems. In the variant, the underlying network differs from the original one due to a different interpretation of flow—see also the remark on page 60 in Section 4.2.4: in the new network a minimum cost flow corresponds to a bend-minimum shape in which, satisfying a second optimization goal, the number of 180-degree angles between edges is maximal. This avoids unnecessary staircase-like structures in the shape and thus reduces the number of segments. We refer to the algorithm in (Tamassia, 1987) as the *traditional method* and to its variant as the *segment-saving method*. We present the experimental results for the segment-saving method and report the cases for which the traditional method shows a different behavior.

   We replace bends by artificial vertices in the resulting orthogonal representation and get a simple orthogonal representation H'_4 that we use as the input for the compaction algorithms. The number of vertices in H'_4 is the number of vertices in the original graph plus the number of artificial vertices, which represent crossings, plus the number of artificial vertices, which represent bends.

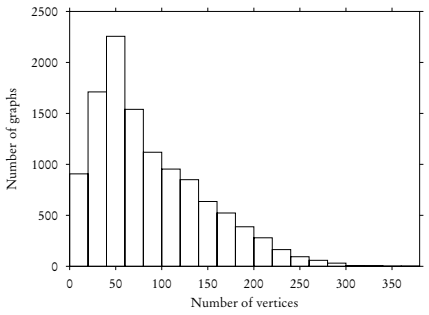
We choose these strategies, since they are among the orthogonal methods which yield the best layouts in practice, see (Di Battista *et al.*, 1997) for an evaluation. Figures 4.25(a), (b) and (c) show the distribution of instance sizes for the different test-suites. The data is not equally distributed since not only the number of vertices determines the size of an instance for the compaction problem, but also the number of crossings and bends which

[11] For the four-planar biconnected graphs, the quasi-trees and the planar graphs in the test-suite there is nothing to do in this step; we just set $G' = G$.

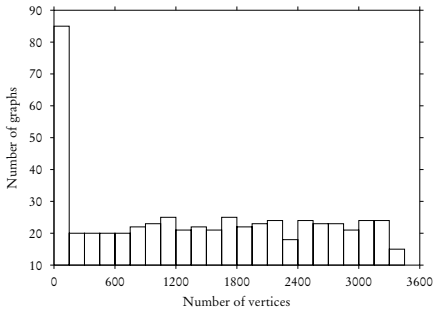
are introduced during the conversion from the original graph G to the simple orthogonal representation H_4'. The peak in Figure 4.25(c) results from the larger number of smaller quasi-trees. Figure 4.25(d) shows the number of crossings that the method described above inserts into the real-world graphs; the other test-suites contain planar graphs.



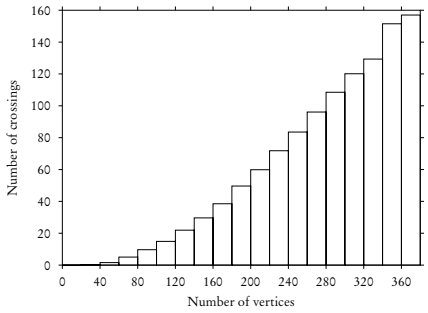
(a) Biconnected four-planar graphs: size



(b) Real-world graphs: size



(c) Quasi-trees: size



(d) Real-world graphs: crossings

Figure 4.25: Distribution of graphs in the test-suites. The size refers to the number of vertices in the simple orthogonal representation H_4', *i.e.*, after planarization, orthogonalization and replacement of bends

We run the implementations of the 14 compaction techniques under evaluation on the two simple orthogonal representations of each graph in the test-suite resulting from applying the traditional and the segment-saving method. We use a Sun Enterprise 450 with 1 GB of main memory and two 400 MHz-CPUs for the experiments.

4.4.3 Computational Results

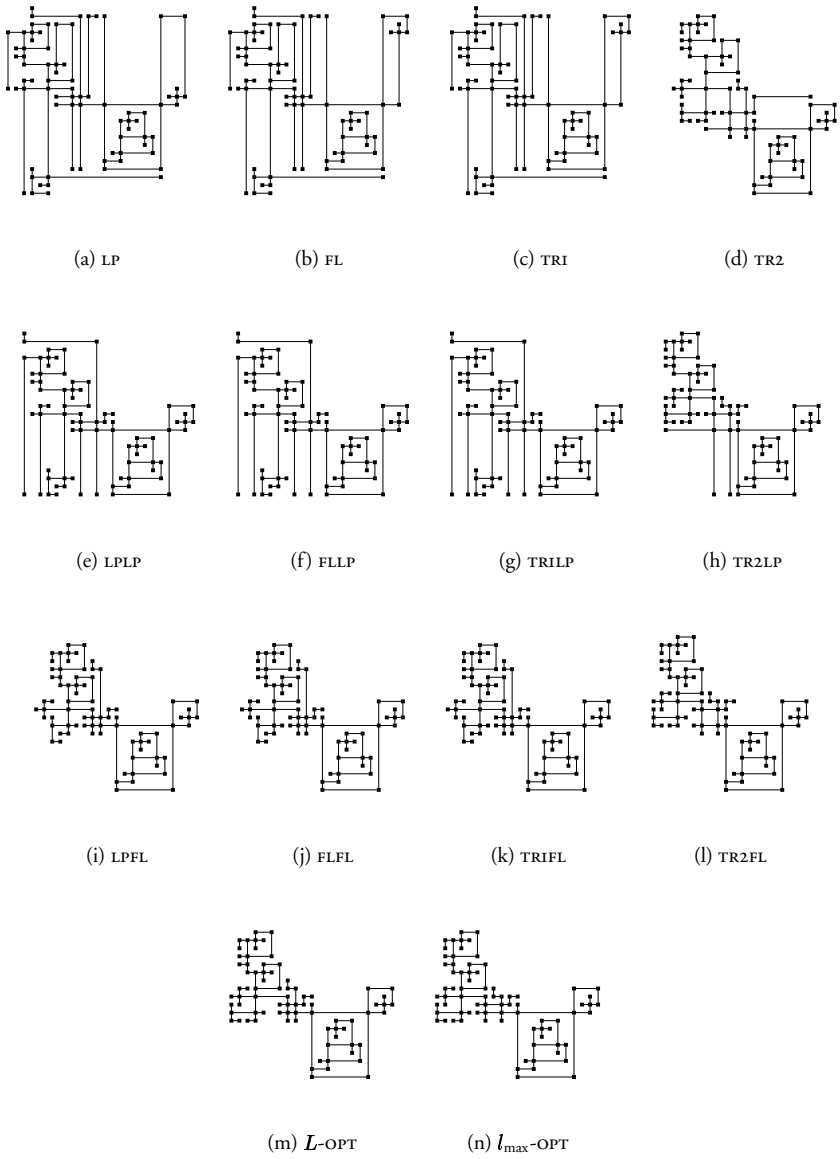
Prefacing the data evaluation of all graphs, we present in Table 4.3 and Figures 4.26 and 4.27 detailed results for the selected example `quasitree.60.6.lgr` from the collection of small quasi-trees for both the traditional GIOTTO method and the segment-saving

strategy. The instance illustrates many of the characteristics we observe when running the 14 compaction algorithms on the set of more than 12,500 graphs from all test-suites, and the pictures support a better understanding of many of the phenomena that can be observed in the computational results.

Method	$\mathbf{W}(\Gamma)$	$\mathbf{H}(\Gamma)$	$\mathbf{P}(\Gamma)$	$\mathbf{A}(\Gamma)$	$\mathbf{L}(\Gamma)$	$\mathbf{l}_{\max}(\Gamma)$	$\mathbf{t}_{\text{comp}}/\text{s}$
traditional							
LP	22	23	45	506	265	14	0.02
FL	22	23	45	506	263	14	0.03
TRI	22	23	45	504	263	14	0.05
TR2	20	22	42	440	170	8	0.04
LPLP	21	20	41	420	213	17	0.12
FLLP	21	20	41	420	213	17	0.13
TRILP	21	20	41	420	213	17	0.13
TR2LP	19	20	39	380	170	8	0.04
LPFL	20	18	38	360	152	8	0.17
FLFL	20	18	38	360	152	8	0.18
TRIFL	20	18	38	360	152	8	0.17
TR2FL	19	19	38	361	149	8	0.17
L-OPT	20	18	38	360	148	8	7.08
$l_{\max}$-OPT	20	18	38	360	148	8	18.97
segment-saving							
LP	26	26	52	676	238	13	0.01
FL	26	26	52	676	234	13	0.03
TRI	26	26	52	676	234	13	0.02
TR2	25	24	49	600	322	24	0.04
LPLP	24	23	47	552	245	16	0.04
FLLP	24	23	47	552	245	16	0.04
TRILP	24	23	47	552	245	16	0.03
TR2LP	24	22	46	528	275	23	0.10
LPFL	22	22	44	484	173	7	0.16
FLFL	22	22	44	484	173	7	0.16
TRIFL	22	22	44	484	173	7	0.16
TR2FL	23	22	45	506	239	22	0.21
L-OPT	24	13	37	312	161	7	97.15
$l_{\max}$-OPT	22	13	35	286	161	7	276.7

Table 4.3: Detailed results for `quasitree.60.6.lgr`, see also Figures 4.26 and 4.27. The last column contains the CPU-time t_{comp}

We provide a detailed discussion of the results for the important aesthetic criteria total edge length and running time. As we will see, the data for the other criteria look similar, and good values for total edge length also lead to good values with regard to the other criteria.



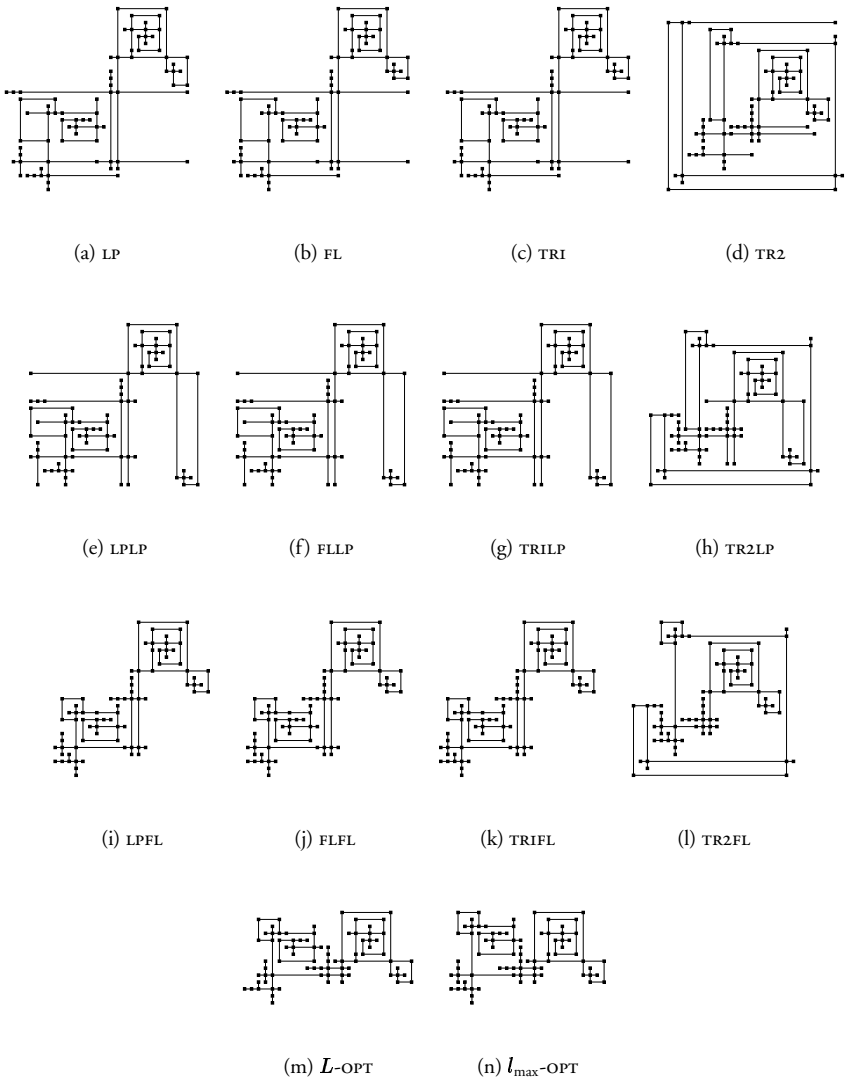
(a) LP (b) FL (c) TRI (d) TR2

(e) LPLP (f) FLLP (g) TRILP (h) TR2LP

(i) LPFL (j) FLFL (k) TRIFL (l) TR2FL

(m) L-OPT (n) $l_{\max}$-OPT

Figure 4.26: Orthogonal drawings of the instance `quasiTree.60.6.lgr` resulting from the 14 evaluated compaction techniques, shape computed with the traditional GIOTTO-algorithm. See also Table 4.3 and Figure 4.27



(a) LP (b) FL (c) TRI (d) TR2

(e) LPLP (f) FLLP (g) TRILP (h) TR2LP

(i) LPFL (j) FLFL (k) TRIFL (l) TR2FL

(m) L-OPT (n) $l_{\max}$-OPT

Figure 4.27: Orthogonal drawings of the instance `quasiTree.60.6.lgr` resulting from the 14 evaluated compaction techniques, shape computed with the segment-saving GIOTTO-algorithm. See also Table 4.3 and Figure 4.26

Total Edge Length

First we consider the set of biconnected four-planar graphs. We divide the instances in subgroups according to their sizes with steps of 20 vertices. For each group we compute the average total edge length: first for the four constructive heuristics (see Figure 4.28), then for the four methods with longest path-based improvement (undisplayed) and last for the methods with flow-based improvement (Figure 4.29). We show the results relative to the optimum total edge length which we compute with L-OPT. Longest path-based post-compaction does not lead to significant improvements in terms of total edge length and sometimes changes the drawings for the worse, thus it hardly justifies its categorization as improvement heuristics. This behavior, identifiable also in Figures 4.26(e)-(h) and especially 4.27(e)-(h), is due to the fact that the method places all vertices to the leftmost and bottommost possible position and can also be observed with the practical graphs and the quasi-trees. We therefore will not display the data for post-compaction with the longest path method.

It is obvious that the biconnected graphs are easy to compact: Already the worst constructive method in terms of edge length, LP, achieves quite good results. Improving the drawings with the flow method often results in an optimal drawing. The plot also indicates that the methods LPFL, TRIFL and FLFL produce quite similar-looking drawings—this is also true for the other test-suites. Here, the reason is that the dissections are equal or similar and the computation of the assignment is very constrained.

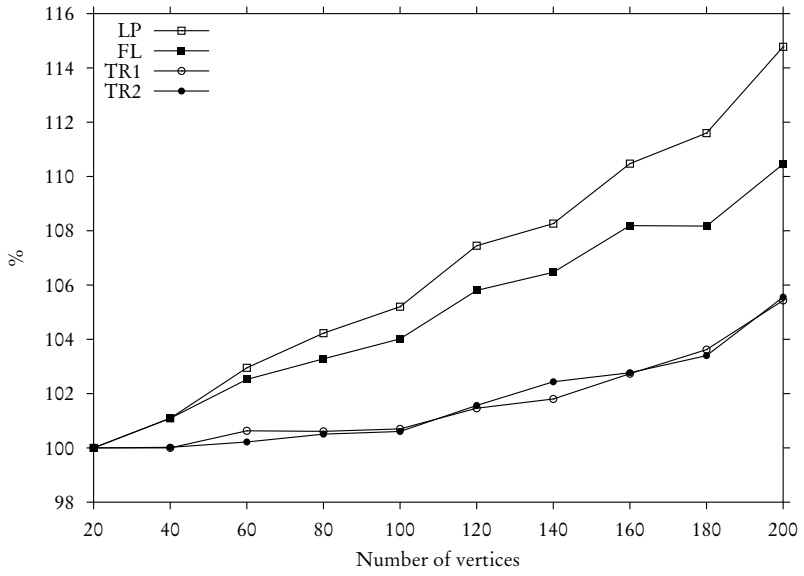


Figure 4.28: Four-planar biconnected graphs: total edge length relative to optimal value. Constructive heuristics

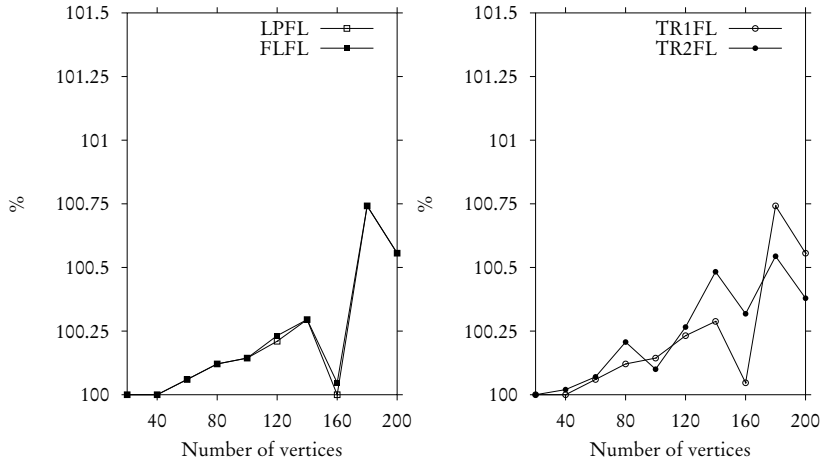


Figure 4.29: Four-planar biconnected graphs: total edge length relative to optimal value. Improvement with flow

We partition the 11,582 graphs corresponding to practical data in the same way as the biconnected graphs. Figure 4.30 shows the resulting total edge length for different methods relative to the optimum value computed by L-OPT. On the one hand, the practical graphs behave like the biconnected graphs: The heuristics are close to the optimum value and almost reach it when using improvement with flow. On the other hand, larger instances are easier to compact, whereas the biconnected graphs indicate the opposite. This is due to the high number of crossings resulting from the prior planarization step (see Figure 4.25(d) for the distribution of crossings). The higher this number, the simpler the shapes of the faces. For the larger graphs, we often observe that almost all faces have a rectangular shape; this explains the good performance of all methods for big planarized graphs.

A different view of the improvement by computing Δ-minimal assignments shows the influence of the flow method on the quality more drastically. In Figure 4.31, we group the graphs according to their size using a step size of 50 vertices and show for each constructive heuristics its value before and after the improvement step. Again, it can be observed that the big planarized graphs are almost optimally compacted by the heuristics. Additionally, the plot illustrates that the choice of the constructive heuristics in the first step does not have a big impact on the final quality: what matters is the improvement with the flow method.

The most challenging instances for the compaction problem are the quasi-trees. We first consider the smaller instances where L-OPT can find the optimal solution or at least a very good bound within the time limit. These are 75 graphs with 40 to 85 vertices in the original graph, resulting in 44 to 121 vertices in the simple orthogonal representation. Figure 4.32 illustrates the quality of the heuristics with respect to the optimum value. Again, the best heuristics perform very well: Even for these hard instances, four of them

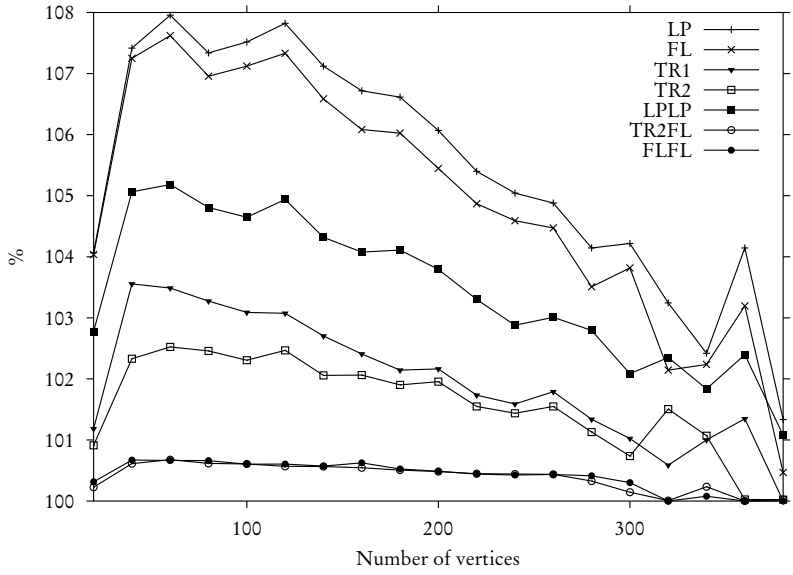


Figure 4.30: Practical graphs: total edge length relative to optimal value

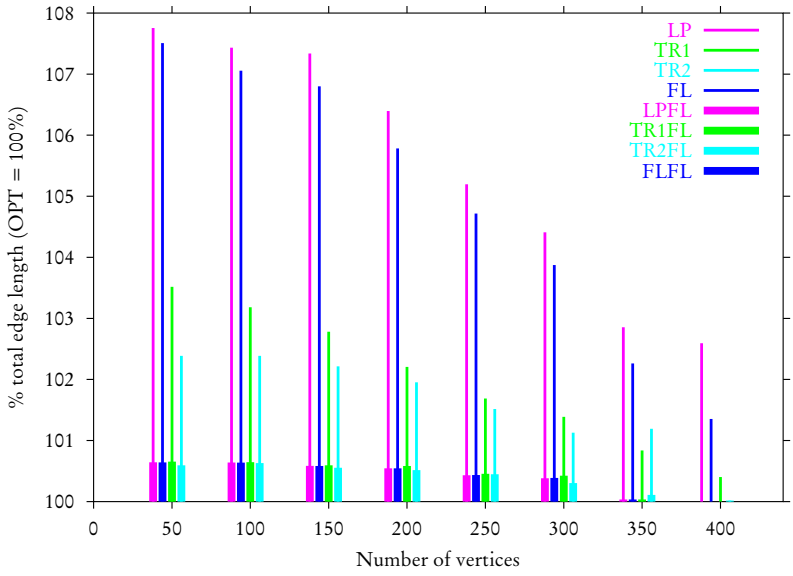


Figure 4.31: Practical graphs: impact of the flow method

(TRIFL, TR2FL, LPFL, and FLFL) are never more than 10% off the optimum value, alternating at the top position among the heuristic methods. In the plot, we only show TRIFL and FLFL, the other two are very close to FLFL. A further observation is that the quality of the methods is relatively independent of the graph size.

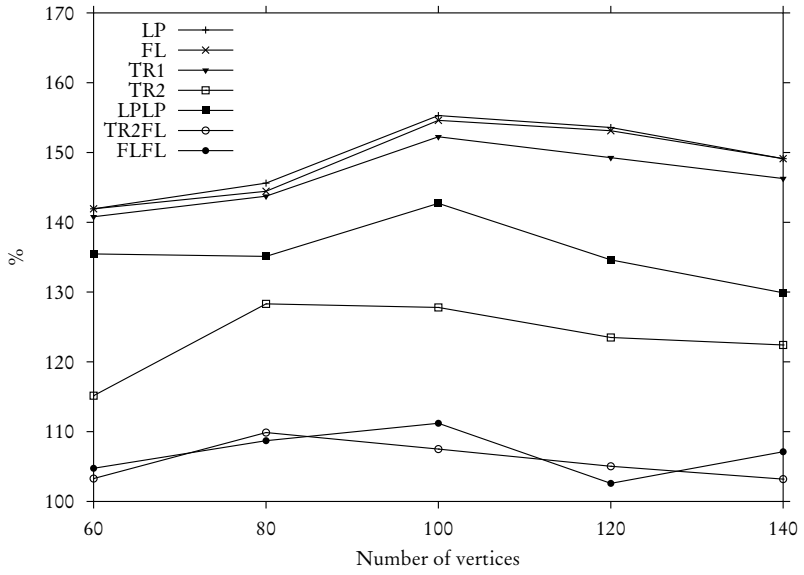
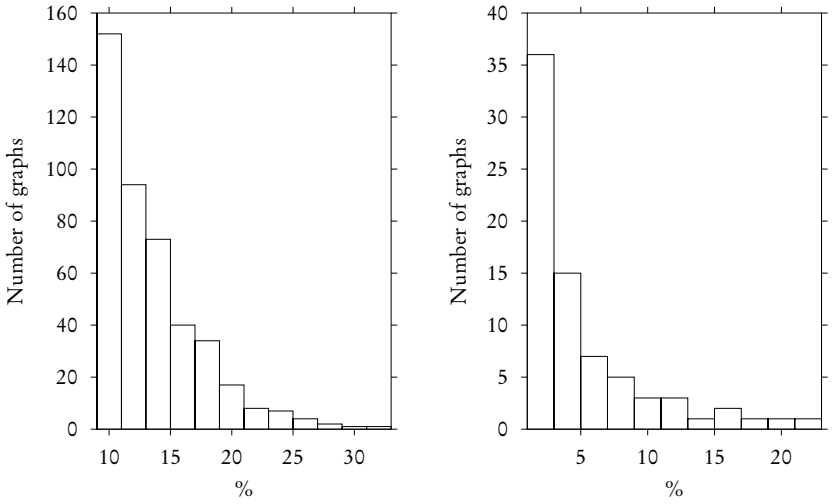


Figure 4.32: Small quasi-trees: total edge length relative to optimal value

As Figures 4.29-4.32 indicate, the combined constructive and improvement heuristics compute almost optimal solutions for instances of all graph classes. However, the plots show average values in each subgroup. What the figures hide is that some instances exist for which the quality difference between the heuristics and the exact algorithm is quite remarkable. Therefore, we take a more careful look at the comparison with the exact algorithm L-opt and compare its solution of each instance to the value of the best heuristics for that instance. We display the data for the practical graphs and the quasi-trees in Figure 4.33; no anomalies exist for the biconnected graphs. The plots show the number of graphs that fall in certain quality categories. For most of the practical graphs, the quality difference between optimally compacted drawings and those obtained by the best heuristics is less than ten percent. But Figure 4.33(a) demonstrates that this is not the case for all graphs: many instances exist for which the usage of an exact algorithm really pays off. The biggest difference is 32%. Figure 4.33(b) shows the deviations for the small quasi-trees. Here, for most of the graphs the heuristics come close to the optimum, nevertheless, for more than one third of the instances the difference is greater than five percent.

Finally, we look at the trend when the sizes of the quasi-trees grow and investigate the quality of the methods for 490 larger instances in the range of 100 to 2,500 original vertices. Here, we choose TR2FL as the comparison method, see Figure 4.34. Again, TRIFL,



(a) Practical graphs (b) Small quasi-trees

Figure 4.33: Deviation of optimal total edge length in quality categories

FLFL and LPFL are very close together and manage in some cases to beat the comparison method. In the plot, we display LPFL and the four constructive heuristics whose quality decreases as the instance sizes grow. It can be seen that the methods based on rectangular dissection perform similarly, as a stand-alone heuristics TR2 is the best. But using flow compaction as an improvement almost nullifies this advantage.

Running Time

All implementations run very fast on the biconnected and practical instances. For the biconnected graphs all heuristics stay below 0.4 seconds and L-OPT stays below 1.12 seconds. Figure 4.35 displays the running times for the practical test-suite. The constructive heuristics take at most half a second and the improvement heuristics take at most 0.8 seconds for all instances. Even L-OPT, which computes a provable optimal solution, stays below 4 seconds on all instances but the big quasi-trees—note that the values in the plot are again average values of graphs within one subgroup of similar size. A more detailed look at the running time behavior of the exact algorithm provides Figure 4.36. The longest running time needed for a real-world example is 9.89 seconds. We find it remarkable that the exact algorithm solves almost all instances to optimality within short computation time.

Figure 4.37 illustrates the time the heuristics need for the hard compaction instances. Here, most methods stay inside a 5 second time limit for graphs up to 1,000 vertices. On the large instances, the running time increases to up to 75 seconds for the flow improvement heuristics. Generally, there is a typical pattern for the performance of the methods

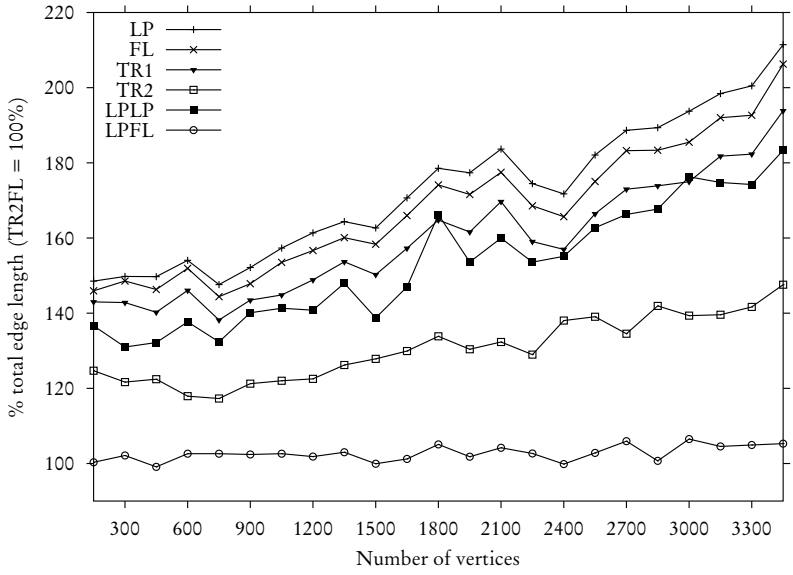


Figure 4.34: Big quasi-trees: total edge length relative to TR2FL

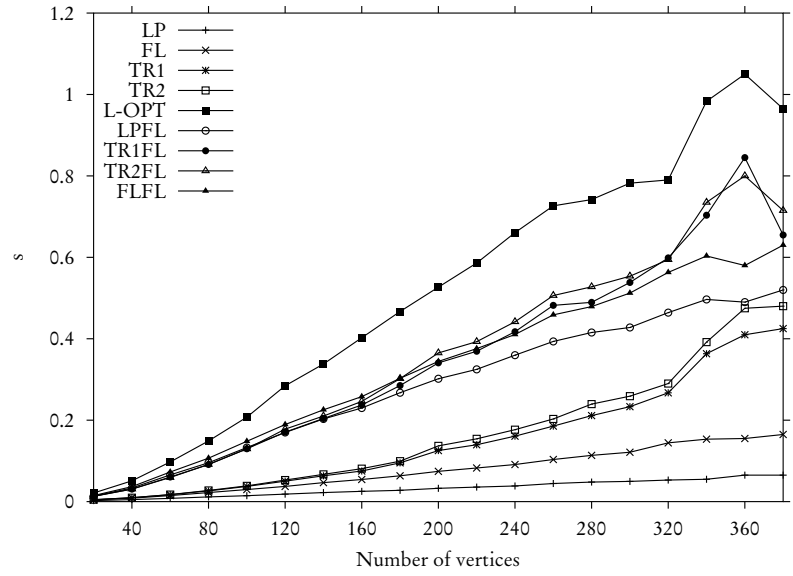
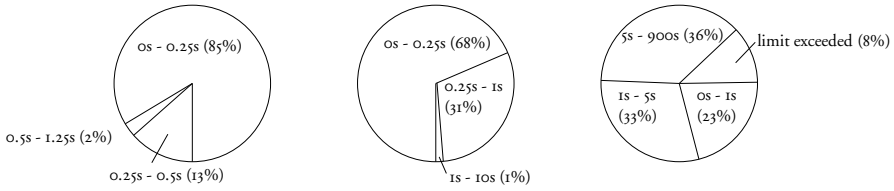


Figure 4.35: Practical graphs: running time



(a) Biconnected graphs (b) Practical graphs (c) Small quasi-trees

Figure 4.36: Running time behavior of *L*-OPT

which can be observed in all graph sets: Heuristics LP beats every other implementation in terms of running time. For the flow-based methods, FL has an advantage over TR1 and TR2, which are the slowest among the constructive methods due to a more complex dissection process. This order remains the same when applying an improvement heuristics.

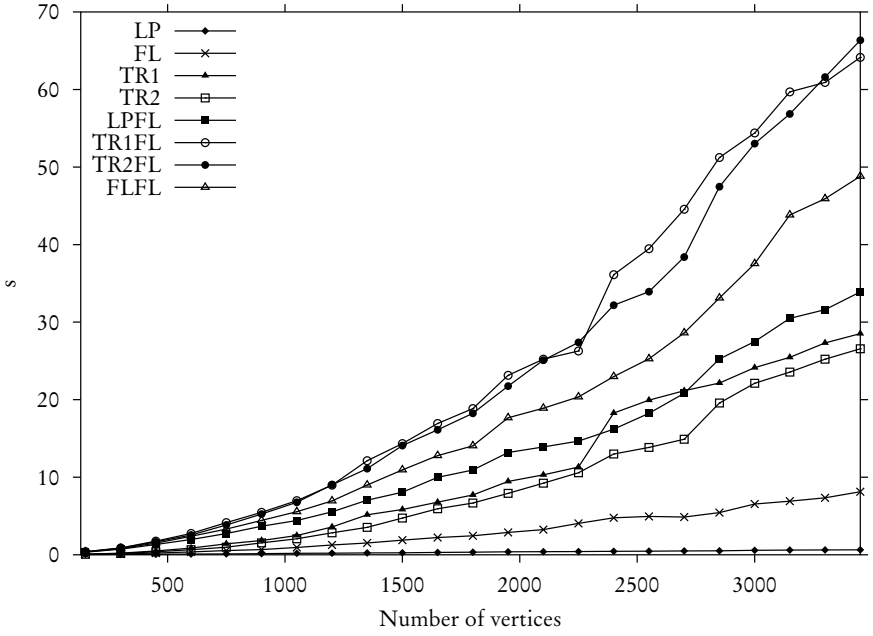


Figure 4.37: Quasi-trees: running time

Other Criteria

The maximal edge lengths and the area in the computed drawings behave very similar to the total edge lengths for the given graph sets and we do not present the detailed

data here.[12] For the biconnected and practical graphs, the average maximal edge lengths per subgroup lie close together. Among the constructive methods, TRI and TR2 again have a slight advantage over the other heuristics. As for the total edge length, the flow improvement equalizes the results. The results for the quasi-trees, however, show a bigger variation. Heuristics TR2 performs best among the constructive methods, but combination with the flow improvement can drastically reduce the lengths for all constructive methods. Here, too, the flow improvement dominates the quality of the result, the differences in the results of the initial methods vanish. The area and half-perimeter results show the same behavior: In case of hard instances, flow improvement can help to enhance the quality and to close the gap created by the initial methods, otherwise the constructive heuristics lie close together and show only a slight improvement when flow is applied on their initial layout.

In the following, we refer to the different test-suites as BICO, PRAC, and QUASI and also refer to the small and large instances of QUASI as SQUASI and LQUASI, respectively. We count the number of iterations in the one-dimensional compaction scheme depending on the variant of the algorithm and the improvement heuristics. Table 4.4 shows the respective maximum and average values. It is surprising that even the maximum numbers are low.

	BICO	PRAC	SQUASI	LQUASI
segment-saving GIOTTO				
LPFL	3 (1.63)	4 (1.98)	4 (2.70)	5 (3.22)
TRIFL	3 (1.19)	4 (1.52)	4 (2.70)	5 (3.26)
TR2FL	3 (1.22)	4 (1.52)	4 (2.49)	6 (3.26)
FLFL	3 (1.60)	4 (1.97)	4 (2.70)	6 (3.25)
traditional GIOTTO				
LPFL	4 (1.94)	5 (2.17)	3 (2.85)	6 (3.30)
TRIFL	4 (1.64)	5 (1.93)	3 (2.85)	5 (3.29)
TR2FL	4 (1.64)	4 (1.79)	4 (2.52)	7 (3.30)
FLFL	4 (1.91)	5 (2.17)	3 (2.85)	6 (3.28)

Table 4.4: Maximum number of iterations in the one-dimensional compaction scheme, average values in parentheses (see also Algorithm 4.3 on page 62)

Turn-Regularity

One of the motivations to use the concept of turn-regularity within a constructive heuristics for compaction problems is to decrease the number of artificial vertices and edges which are inserted during that process. We count the numbers of inserted edges for the rectangular dissection method (RD) as well as for the two turn-regularity-based dissection techniques (TD1 and TD2) and present the results in Table 4.5. The number of artificial

[12] The full data of the experimental study is available at `http://www.ads.tuwien.ac.at/compaction`.

edges is significantly lower in the dissection strategies which use the concept of turn-regularity. For the biconnected graphs, the rectangle-based method RD inserts up to 13 and 36 times more edges as compared to TD1 and TD2, respectively. In the case of harder instances the factors decrease, but still roughly up to two and ten times as many edges are needed to fully dissect the faces in rectangle-shaped subfaces.

Our experimental results, however, show that the influence of the number of artificial vertices and edges is not as dramatic as Table 4.5 suggests. Even when inserting a small number of artificial edges, local decisions have to be made which may have a strong impact on the overall drawing.

	BICO	PRAC	SQUASI	LQUASI
	segment-saving GIOTTO			
RD	17.7%	24.5%	52.8%	36.6%
TD1	1.4%	3.7%	32.1%	16.6%
TD2	0.5%	1.0%	7.7%	3.7%
	traditional GIOTTO			
RD	30.4%	31.3%	60.3%	37.6%
TD1	8.0%	7.7%	40.7%	17.3%
TD2	2.3%	2.1%	10.6%	3.9%

Table 4.5: Percentage of inserted artificial vertices in the dissection process related to the total number of edges

Bridgeman *et al.* (2000) report a ratio of turn-regular faces of about 89% in their test-suite of four-planar biconnected graphs. We have the same results with exception of the small quasi-trees, where only about 81% of the faces are regular. In general, however, the irregular faces are the larger ones which count as much in that measure as small faces which are likely to have rectangular shape.[13] Furthermore, in many cases one of the irregular faces is the external face which can have a high impact on the complexity of the compaction problems.

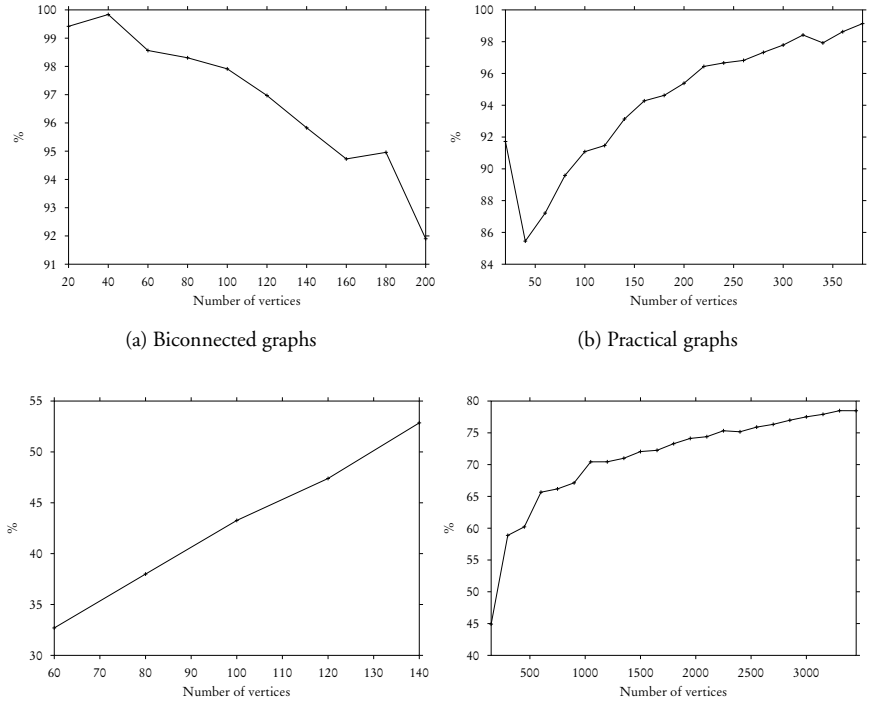
Another indicator that orthogonal representations of biconnected graphs and the practical graphs in the test-suite are easy to compact is that many of the graphs are turn-regular, that is, all of their faces are turn-regular. Table 4.6 shows how many representations have this property within a test-suite, depending on the variant of GIOTTO. It can be observed that about twice as much biconnected representations are turn-regular than representations from the practical test-suite; none of the quasi-trees is turn-regular. The segment-saving strategy more than doubles the percentage of turn-regular instances.

Figure 4.38 shows a more suitable measure of turn-regularity: the number of edges in turn-regular faces as compared to the total number of edges.

[13] Note that the smallest possible irregular faces are bounded by eight edges and look like Figure 4.21(e) on page 76.

	BICO	PRAC	QUASI
traditional GIOTTO	31.6%	15.7%	0%
segment-saving GIOTTO	71.8%	36.9%	0%

Table 4.6: Percentage of turn-regular representations in the test-suite



(a) Biconnected graphs

(b) Practical graphs

(c) Small quasi-trees

(d) Big quasi-trees

Figure 4.38: Percentage of edges in turn-regular faces, compared to total number of edges

4.4.4 Conclusions

The experimental study presented in Sections 4.4.1 to 4.4.3 evaluates the state-of-the-art compaction techniques for orthogonal graph drawing in terms of quality and run time performance by running 14 implementations on three fundamentally different test-suites. At this point, we summarize the conclusions we draw from the extensive experimental results.

Heuristics perform very well on most instances of the compaction problem. Especially for the data from the easy and practical instances we can observe an excellent behavior, both in terms of quality and running time. We want to emphasize, however, that in some

cases it is desirable to get an optimal drawing, *e.g.*, when quality is more important than running time. There are cases in which it is worth spending an extra amount of running time in order to produce a high-quality drawing. The test-suites contain many examples (especially real-world graphs) for which the quality difference between the optimal solution and the best heuristics is more than 15 percent. Moreover, the implementation of the integer linear programming-based algorithm is necessary to measure the quality of the heuristics.

Graphs with many crossings are easy to compact. The topology-shape-metrics scheme treats crossings as artificial vertices. The more crossings a drawing contains, the simpler are the shapes of its faces. For the big planarized graphs in the practical test-suite we observe that in many cases almost all faces have rectangular shape. Such graphs provide easy instances of the compaction problem as shown in Section 4.2.4.

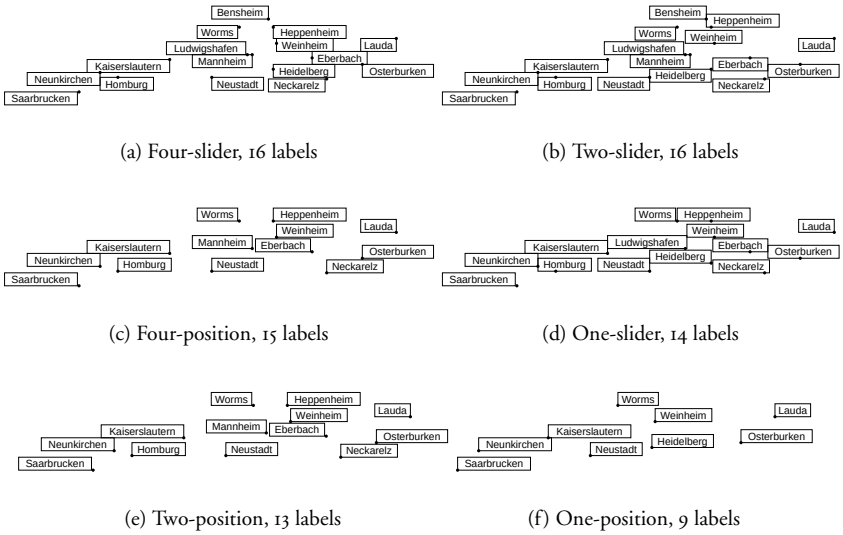
Longest-path assignment suffices for constructive heuristics. Computing minimal assignments is much faster than Δ-minimal assignments (see Chapter 3). The structure of the auxiliary representations used in the constructive heuristics is very special which leads to the fact that minimal and Δ-minimal assignments are almost identical—see also Remark 4.3 on page 58. We recommend to use the faster longest-path methods within the constructive heuristics. More important, however, is the following issue.

The choice of the constructive heuristics does not matter as long as flow compaction is used as post-processing. Although the results for the constructive methods differ significantly, the differences vanish when using the flow-based improvement heuristics based on Δ-minimal assignments. Therefore, we propose to use a simple constructive method, *e.g.*, rectangular dissection and minimal coordinate assignment, followed by a flow-based post-processing step, which yields also the best running time among the good heuristics. The implementational effort for this variant is low as compared to the relatively complex turn-regularity-based techniques.

Chapter 5 Map Labeling

> Then came in all the king's wise men: but they could not read the writing, nor make known to the king the interpretation thereof.
>
> Daniel 5:8

In this chapter, we will present algorithms for the different labeling problems that work in any of the six labeling models displayed in Figure 1.3 on page 10 of the introduction. We allow several labels per point-feature and labels of different sizes. Figure 5.1 shows provably optimal labelings of a part of a practical instance for the label number maximization problem under the different rectangular labeling models.



(a) Four-slider, 16 labels

(b) Two-slider, 16 labels

(c) Four-position, 15 labels

(d) One-slider, 14 labels

(e) Two-position, 13 labels

(f) One-position, 9 labels

Figure 5.1: Provably optimal solutions for the label number maximization problem in the six different axis-parallel rectangular labeling models for a part of the German railway instance

Section 5.1 contains precise mathematical definitions of the labeling problems and mentions previous work that aims at solving these problems in the different models. In particular, we will provide a comprehensive complexity overview of labeling problems; most of them are *NP*-hard optimization problems, but some restricted cases can be solved in polynomial time.

In Section 5.2 we present our new combinatorial approach to the problems. We proceed in a similar fashion as for the compaction problems in the previous chapter. We define two constraint graphs, the *labeling graphs* for an instance of a labeling problem, and require several graph-theoretic properties to be satisfied in order to use this pair of graphs to compute a solution of the labeling instance. As for compaction, we determine the placement by computing assignments in the underlying constraint graphs. Again, we reformulate the original problems as combinatorial versions and prove equivalence.

We describe translations of the equivalent combinatorial problems into integer linear programming formulations in Section 5.3 and show how to characterize the set of feasible solutions of the pure labeling problem as the convex hull of a zero-one polytope. We present classes of inequalities and integrality constraints that describe the integral points in this *labeling polytope*. As it will turn out, violated inequalities in one of these classes, the *positive cycle inequalities*, are difficult to find, since the decision version of the corresponding separation problem is *NP*-complete. We derive an alternative formulation without the positive cycle inequalities but at the cost of additional variables. The characterization by inequalities enables us to develop integer linear programming formulations for many of the labeling problems. Our main focus is on the label number maximization problem, but we also demonstrate how to model the other problem types with integer linear programs.

In Section 5.4, we describe new exact algorithms for the labeling problems in the six axis-parallel rectilinear labeling models. We present an iterative branch-and-bound based technique and a branch-and-cut algorithm which solve instances of the labeling problems to optimality.

Of course, our algorithms need exponential time in the worst case. Nevertheless, Section 5.5 shows that they are competitive for practical examples. We evaluate our implementation for the label number maximization problem using a large number of real-world instances and a widely used generator for benchmark instances.

5.1 Labeling Problems

This section contains precise characterizations of different point-feature map labeling problems in the six different axis-parallel labeling models. We provide a general definition for instances of labeling problems which allows an unbounded number of labels per point-feature and labels of different size. We introduce five kinds of labeling problems which differ in whether they allow omitting labels, scaling, or overlaps. The *labeling decision problem* asks whether an unscaled labeling without overlaps exists. A solution of the *labeling problem* consists of such a labeling, if one exists. Since in practice it is quite unsatisfactory, in case the answer to the first two problems is "no", three optimization problems arise: The *label number maximization problem* asks for an overlap-free, unscaled labeling for a largest subset of the labels. All labels must be placed in the *label size maximization problem*, in which the task is to determine the maximum scale factor and a corresponding labeling without overlaps. The *label overlap minimization problem* tolerates overlaps but asks for an unscaled labeling for all labels with the minimum number of overlaps.

All of the above problems are *NP*-complete or *NP*-hard for many of the labeling models. At the end of this section we provide a complexity overview of the most popular problems in the different labeling models.

We start by defining some notation about rectangles in the plane: Let p be a point in the plane; we also refer to p by its two coordinates $(p_x, p_y) \in \mathbb{Q}^2$. Let $\mathbf{R}$ be the set of axis-parallel iso-oriented rectangles in the plane. A rectangle $R \in \mathbf{R}$ is characterized by two points: the lower left corner $ll(R)$ and the upper right corner $ur(R)$. If $ll(R)$ and $ur(R)$ differ in both, x- and y-coordinate, we call R *non-trivial*. Since R is iso-oriented, the upper left corner $ul(R) = \big(ll(R)_x, ur(R)_y\big)$ and the lower right corner $lr(R) = \big(ur(R)_x, ll(R)_y\big)$ are uniquely defined. The *boundary* of R is given by the four line segments

$$\begin{aligned} l(R) &= \big(ll(R), ul(R)\big) && \text{(left boundary)} \\ u(R) &= \big(ul(R), ur(R)\big) && \text{(top boundary)} \\ r(R) &= \big(ur(R), lr(R)\big) && \text{(right boundary)} \\ b(R) &= \big(lr(R), ll(R)\big) && \text{(bottom boundary)} \ . \end{aligned}$$

Figure 5.2 illustrates these definitions. The *open intersection* $S = Q \cap R$ of two rectangles Q and R is non-empty if and only if S is a non-trivial rectangle, *i.e.*, both width and height of S must be strictly greater than zero. Hence, two rectangles may touch and do not intersect.

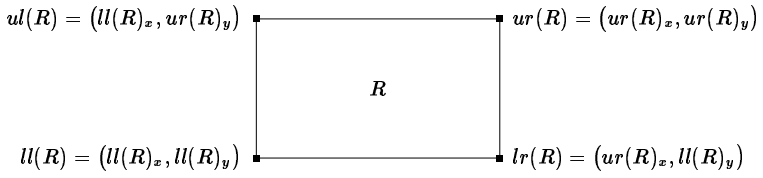


Figure 5.2: Rectangle $R \in \mathbf{R}$

We present a formal definition of an instance of a labeling problem which is quite general: We allow different labels to have different sizes and impose no restriction on the number of labels per point. An instance is then just a set of objects, the labels, and three functions w, h, and a. The first two functions describe the width and height of the rectangle that represents the appropriate label, the third function denotes the point in the plane which receives the label.

Definition 5.1 (Instance of a labeling problem). An instance of a labeling problem is a four-tuple

$$\big(\Lambda, w, h, a\big)$$

where $\Lambda = \{\lambda_1, \ldots, \lambda_l\}$ is a set of l labels, and w, h, and a are functions from the set of labels to positive numbers or points, respectively. More precisely, $w : \Lambda \to \mathbb{Q}_+$ and $h : \Lambda \to \mathbb{Q}_+$ are the *widths* and *heights* of the labels, and $a : \Lambda \to \mathbb{Q}^2$ denotes the points that receive the labels.

The common aim of all labeling problems is to find a placement of all or many labels. We define a *labeling* as a placement that respects the three functions given by the labeling instance. The requirement how the rectangles have to be placed with respect to the points receiving the labels influences the labeling model.

Definition 5.2 (Labeling). Given an instance of a labeling problem (Λ, w, h, a). A *labeling* according to a *labeling-model* is a function of labels to rectangles $\rho : \Lambda \to \mathbf{R}$, so that the following conditions hold:

(L1) Rectangle $\rho(\lambda)$ has width $w(\lambda)$ and height $h(\lambda)$ for every $\lambda \in \Lambda$.

(L2) Function ρ places all rectangles according to the labeling model, that is,

$$a(\lambda) \in \begin{cases} l(\rho(\lambda)) \cup u(\rho(\lambda)) \cup r(\rho(\lambda)) \cup b(\rho(\lambda)) & \text{in the four-slider model} \\ t(\rho(\lambda)) \cup b(\rho(\lambda)) & \text{in the two-slider model} \\ b(\rho(\lambda)) & \text{in the one-slider model} \\ \{ll(\rho(\lambda)), ul(\rho(\lambda)), ur(\rho(\lambda)), lr(\rho(\lambda))\} & \text{in the four-position model} \\ \{ll(\rho(\lambda)), lr(\rho(\lambda))\} & \text{in the two-position model} \\ \{ll(\rho(\lambda))\} & \text{in the one-position model} \end{cases}$$

Definition 5.2 does not state anything about the overlaps that might occur among the rectangles. In all labeling problems, overlaps are either forbidden or determine the quality of a solution. We provide a separate definition of the set of overlapping labels:

Definition 5.3 (Overlaps of a labeling). Let ρ be a labeling for an instance (Λ, w, h, a). Two distinct labels $\lambda, \mu \in \Lambda$ *overlap* if the open intersection $\rho(\lambda) \cap \rho(\mu)$ is not empty. The overlaps of a labeling is the set

$$O(\rho) = \left\{ \{\lambda, \mu\} \in \binom{\Lambda}{2} \mid \rho(\lambda) \cap \rho(\mu) \neq \varnothing \right\} .$$

A first question is whether a labeling without overlaps exists. This gives rise to the following decision problem.

Definition 5.4 (Labeling decision problem, LAB-D). Given an instance $I = (\Lambda, w, h, a)$ of a labeling problem, does a labeling ρ for I exist with $O(\rho) = \varnothing$, *i.e.*, a labeling in which the number of overlaps is zero?

Closely related to the decision problem is the *labeling problem* which asks for an actual labeling, if one exists.

Definition 5.5 (Labeling problem, LAB). Given an instance $I = (\Lambda, w, h, a)$ of a labeling problem, find a labeling ρ for I with $O(\rho) = \varnothing$, if one exists.

For practical instances, the labeling problem is often infeasible. This gives rise to the following three optimization problems. The *label number maximization problem* asks for

a labeling for a largest subset of labels. In certain applications it is necessary to place all the labels; in this case, scaling may be allowed, and we want to find the maximum scale factor which allows a labeling for all labels—this is the *label size maximization problem*. If scaling is not allowed and all labels have to be placed, overlaps cannot be avoided. In this scenario, the number of overlaps must be as small as possible in order to optimize the aesthetic criteria at the beginning of this section. We will refer to this variant as the *label overlap minimization problem*.

Definition 5.6 (Label number maximization problem, LNM).
Given an instance $I = (\Lambda, w, h, a)$ of a labeling problem, find a largest subset $\Lambda_P \subseteq \Lambda$ and a labeling ρ for the instance

$$I' = \left(\Lambda_P, w|_{\Lambda_P}, h|_{\Lambda_P}, a|_{\Lambda_P}\right)$$

with $O(\rho) = \varnothing$. Here, $w|_{\Lambda_P}, h|_{\Lambda_P}, a|_{\Lambda_P}$ are the restrictions of the functions to Λ_P.

Definition 5.7 (Label size maximization problem, LSM).
Given an instance $I = (\Lambda, w, h, a)$ of a labeling problem, find a maximum factor $\alpha \in \mathbb{Q}$ and a labeling ρ for the instance

$$I' = \left(\Lambda, w', h', a\right)$$

with $O(\rho) = \varnothing$. Here,

$$\begin{aligned} w'(\lambda) &= \alpha \cdot w(\lambda) \\ h'(\lambda) &= \alpha \cdot h(\lambda) \ . \end{aligned}$$

Definition 5.8 (Label overlap minimization problem, LOM).
Given an instance $I = (\Lambda, w, h, a)$ of a labeling problem, find a labeling ρ for I so that the number of overlaps $|O(\rho)|$ is minimal.

In the rest of this section, we give an overview of previous work done in the area of point-feature labeling, both for the discrete and the slider models. We refer by n to the number of labels of an instance.

5.1.1 Discrete Models

Most previous work on map labeling concentrates on the discrete models which allow only a finite number of positions per label. While the decision problem LAB-D is *NP*-complete in the four- and the p-position model (for fixed $p \geq 4$) (see Kato and Imai, 1988; Formann and Wagner, 1991; Marks and Shieber, 1991), the labeling problem can be solved in time $\Theta(n \log n)$ via a 2-satisfiability formulation in the two-position model (Formann and Wagner, 1991), and via a simple sweep line algorithm in the one-position model. Iturriaga and Lubiw (1997b) study an interesting variant of the pure labeling problem in the discrete model, the *elastic labeling problem*. They define the rectangles which represent the labels only through their area and allow changing the individual aspect ratios.

The results for the pure labeling problem can directly be extended to the label size maximization problem: LSM is polynomially solvable in the one- and two-position model, and *NP*-hard in the p-position model for fixed $p \geq 4$. For the restricted variant of LSM in the four-position model with labels of unit width and height, Formann and Wagner (1991) present a $\frac{1}{2}$-approximation algorithm which runs in $O(n \log n)$ time. Moreover, they demonstrate that no better approximation factor is possible unless P is equal to *NP*. Kučera, Mehlhorn, Preis, and Schwarzenecker (1993) suggest an $O(4^{\sqrt{n}})$ time algorithm which is able to solve instances with up to 100 labels to provable optimality. The authors exploit the observation that only up to

$$\log\left(4 \cdot \binom{n}{2}\right) = O(\log n)$$

different scale factors are candidates for an optimal solution in the discrete models: In an optimal labeling, either two labels touch, or a label touches another point-feature (otherwise the scale factor could be increased). The observation shows that it suffices to solve $O(\log n)$ decision problems and one labeling problem to solve an instance of the label size maximization problem in the discrete models.

In contrast to the label size maximization problem, the label number maximization problem LNM is also *NP*-hard in the one- and two-position model. Woeginger (2000) shows *NP*-hardness of the two problems by a reduction from the maximum independent set problem in planar cubic graphs which is a known *NP*-hard problem (Garey, Johnson, and Stockmeyer, 1976). *NP*-hardness for LNM in the p-position models follows directly from the *NP*-completeness result for the corresponding decision problems.

Agarwal, van Kreveld, and Suri (1998) study the label number maximization problem LNM in the p-position model for fixed p. They present a $\frac{1}{2}$-approximation algorithm for labels of unit height. The $\Theta(n \log n)$-time algorithm divides the problem into one-dimensional subproblems for which it computes the largest non-overlapping set of intervals by using a greedy strategy. The authors also suggest a polynomial-time approximation scheme in the p-position model when the rectangles have unit height. The algorithm finds a solution of size at least $|\Lambda_P|/(1 + \frac{1}{k})$ for $k \geq 1$ in time $O(n \log n + n^{2k-1})$, where Λ_P is the optimal solution. In case the rectangles differ in their height, the authors suggest a factor $O(\log n)$-approximation algorithm running in time $\Theta(n \log n)$. So far, for this problem no constant factor approximation algorithm is known.

The first exact algorithms for LNM in the p-position model are suggested by Cromley (1986) and Zoraster (1990). Both authors experiment with a zero-one integer linear programming formulation for the maximization problem which they solve approximately using Lagrangian relaxation, subgradient optimization techniques, and several problem-specific heuristics. However, they cannot solve practically relevant instances to provable optimality with their approach.

Verweij and Aardal (1999) suggest the only practically efficient algorithm for computing provably optimal solutions for LNM in the discrete models. They treat the problem as an independent set problem and solve it using a branch-and-cut algorithm. The algorithm is able to optimally label up to 800 point-features using the benchmark generator

from (Christensen, Marks, and Shieber, 1995) within moderate computation time (about 20 minutes) and up to 950 point-features within two hours, as reported in (Verweij, 2000).

Many articles exist on heuristic algorithms for the label number maximization problem and the label size maximization problem (see Wolff and Strijk, 2001). Here, we only mention the paper by Christensen *et al.* (1995). The authors present a comprehensive treatment of the label number maximization problem (LNM) in the four-position model including a comparison of heuristic methods and an extensive computational study. Their procedure for randomly creating labeling instances has become a widely used benchmark generator in the map labeling literature, and we will also use it for our computational experiments in Section 5.5.

5.1.2 Slider Models

Only a few papers exist that focus on continuous models. Hirsch (1982) develops a first algorithmic approach to map labeling problems in the four-slider model. He describes a force-directed method that aims at minimizing the number of label overlaps. Doddi, Marathe, Mirzaian, Moret, and Zhu (1997) consider, among other problems, label size maximization in a continuous model which allows labels to rotate around the point-features. Christensen *et al.* (1995) mention a variant of a continuous model in which circles around a point-feature define the region the appropriate label must touch without intersecting.

We stick to the classification of slider models provided in (van Kreveld, Strijk, and Wolff, 1999), namely into the one-, two-, and four-slider model as illustrated in Figure 1.3. *NP*-completeness of the decision problem LAB-D in the four-slider model is shown independently by van Kreveld *et al.* (1999) and Marks and Shieber (1991), and in the two-slider model by Iturriaga and Lubiw (1997a). The labeling problem in the one-slider model can be solved using a simple greedy sweep line algorithm whereas the corresponding label number maximization problem is *NP*-hard (Woeginger, 2000)[1].

We give an overview of the complexity for the decision problem and the label number maximization problem in Table 5.1. In the cases in which the decision problem LAB-D is *NP*-complete, both the label size and the label number maximization problem are obviously *NP*-hard. It is interesting that all number maximization versions—even in the seemingly simple one-position model—are computationally hard.

Van Kreveld *et al.* (1999) present a $\frac{1}{2}$-approximation algorithm which is able to find a solution of LNM in any of the slider-models with unit height rectangles. The algorithm is a $\Theta(n \log n)$ greedy sweep line algorithm. The sweep line proceeds while repeatedly choosing the label whose right edge is leftmost among all remaining label candidates, if possible. For the same models, the authors suggest a polynomial time approximation scheme. The respective algorithms label at least $(1 - \varepsilon)$ times the optimum number in overall running time $O(n^{4/\varepsilon^2})$.

Strijk and van Kreveld (1999) extend the above mentioned $\frac{1}{2}$-approximation algorithm for the slider models to labels with different heights. For r different label heights, the running time of the algorithm is $O(rn \log n)$.

[1] He uses a similar construction as in his proofs for the discrete models.

Model	Decision/Labeling	Size Maximization	Number Maximization
	Discrete Models		
One-position	$O(n \log n)$ Simple plane sweep	$O(n(\log n)^2)$ binary search on $O(n^2)$ decision problems	*NP*-hard (Woeginger, 2000)
Two-Position	$O(n \log n)$ (Formann and Wagner, 1991)	$O(n(\log n)^2)$ binary search on $O(n^2)$ decision problems	*NP*-hard (Woeginger, 2000)
Four-Position	*NP*-complete (Kato and Imai, 1988; Marks and Shieber, 1991; Formann and Wagner, 1991)	*NP*-hard	*NP*-hard
	Slider Models		
One-Slider	$O(n \log n)$ Greedy sweep line, (Woeginger, 2000)	unknown	*NP*-hard (Woeginger, 2000)
Two-Slider	*NP*-complete (Iturriaga and Lubiw, 1997a)	*NP*-hard	*NP*-hard
Four-Slider	*NP*-complete (van Kreveld *et al.*, 1999; Marks and Shieber, 1991)	*NP*-hard	*NP*-hard

Table 5.1: Complexity overview of the decision problem and the size and number maximization problem for point-feature labeling in different models (n is the number of labels)

An interesting point in (van Kreveld *et al.*, 1999) is the investigation of the relationship between the six axis-parallel rectangular labeling models displayed in Figure 1.3. Given unit square label candidates in a model M_1 and another model M_2, the factor Ψ_{M_1,M_2} denotes how many more points can receive labels in model M_1 than in model M_2, more precisely,

$$\Psi_{M_1,M_2} = \lim_{n\to\infty} \max_{|P|=n} \frac{\text{size(optimal labeling for } P \text{ in } M_1)}{\text{size(optimal labeling for } P \text{ in } M_2)} \ .$$

The authors show that Ψ_{M_1,M_2} is equal to two for many of the relationships. It is worth noting that the factor of the four-slider model compared to the four-position model is between two and four, whereas the factor of the four-slider model compared to the one-slider model is between two and three. Also their computational results on the six models confirm the theoretical result that the slider models allow a placement of a significantly larger number of labels than the discrete models. The four-slider model allows to place up to 15% more labels in real-world instances and up to 92% more labels in pseudo-random instances as compared to the four-position model.

5.2 Combinatorial Characterization

In this section we show how to use constraint graphs in order to solve labeling problems. Like for the compaction problems in graph drawing, we present a reformulation of the original problems in terms of combinatorial properties in the constraint graphs. We will introduce the *labeling graphs*, a pair of constraint graphs which arises from an instance of a labeling problem. The key idea is similar as for the two-dimensional compaction problems: If these graphs satisfy certain properties, we can produce a solution for the original problem by computing two separate feasible assignments. The values in the assignments correspond to x- and y-coordinates of the solution. First, we will show how to guarantee a labeling in the respective labeling model. We achieve this by introducing the *fixed distance arcs*, the *label size arcs*, the *proximity arcs*, and the *boundary arcs*. Then we add the *label separation arcs* for each pair of labels that may possibly overlap. Boundary and label separation arcs belong to the class of *potential arcs*, *i.e.*, we do not add them in advance but have to choose among them to satisfy certain requirements. For each of the problems in Section 5.1, we present a reformulation in terms of requirements to these choices and show that the appropriate combinatorial version is equivalent.

5.2.1 Labeling Graphs

The concept of *labeling graphs* is similar to that of placement graphs in the previous chapter. The labeling graphs are a generic pair of constraint graphs for a map labeling instance. Nodes in the *horizontal labeling graph* D_x correspond to x-coordinates of points and left and right label boundaries, nodes in the *vertical labeling graph* represent y-coordinates of points and bottom and top label boundaries.

Definition 5.9 (Labeling graphs). A pair of *labeling graphs* consists of two constraint graphs $D_x = (N_x, A_x, w_x)$ and $D_y = (N_x, A_x, w_x)$ and corresponds to an instance $I = (\Lambda, w, h, a)$ of a labeling problem. Let $\Pi = \{a(\lambda) \mid \lambda \in \Lambda\}$ be the set of points to be labeled. The node sets of D_x and D_y are

$$N_x = \{x_p \mid p \in \Pi\} \cup \{l_\lambda \mid \lambda \in \Lambda\} \cup \{r_\lambda \mid \lambda \in \Lambda\}$$
$$N_y = \{y_p \mid p \in \Pi\} \cup \{b_\lambda \mid \lambda \in \Lambda\} \cup \{t_\lambda \mid \lambda \in \Lambda\} \ .$$

Here, x_p and y_p are the nodes which represent the x- and y-coordinate of a point-feature, l_λ, r_λ, b_λ and t_λ represent the appropriate coordinates of the left, right, bottom, and top boundary of a label λ. In the style of Chapter 4, we refer to these nodes as the *left*, *right*, *bottom*, and *top limit* of a label. See Figure 5.3.

As for placement graphs, we call a pair of feasible assignments (c_x, c_y) for the labeling graphs D_x and D_y a *coordinate assignment*.

The key idea is the same as for the two-dimensional compaction problems: The actual placement of labels will be performed through computing feasible assignments for the labeling graphs. Each feasible labeling can be produced by assigning values to the nodes of the labeling graphs. The rest of this section deals with identifying properties of labeling graphs which guarantee that feasible assignments for the constraint graphs lead

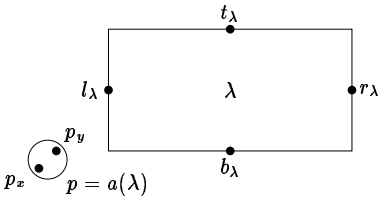


Figure 5.3: Nodes in the labeling graphs corresponding to a label λ and its point-feature $a(\lambda)$

to labelings for the given instance. Furthermore, we show how to control the criteria which must be satisfied by a labeling or influence the quality of a feasible solution in a combinatorial way: Depending on the different labeling problems, this is a combination of number of overlaps, number of placed labels and scale factor. Describing these requirements in a combinatorial manner will build the fundament of integer linear programming formulations for the problems in Section 5.1.

In contrast to the placement graphs introduced in the previous section, labeling graphs have arcs with negative weights as well. For placement graphs, the property of completeness excludes the presence of directed cycles; since the arc weights are non-negative in the constraint graphs for the compaction problems, each cycle corresponds to a cycle of positive weight. This is different for the labeling graphs. Here, many cycles of zero or negative weight occur naturally: A cycle of zero weight corresponds to an equality constraint between two x- or y-coordinates, *e.g.*, the fixed distance between two point-features. Negative weighted cycles represent a relative placement within a certain interval, we will use them to ensure proximity of a label to its point-feature.

In the following, we describe certain arc sets which model parts of the labeling problems. First, we fix the coordinates of the point-features with the *fixed distance arcs*. Then, we model the labels: *Label size arcs* describe the corresponding rectangle of a label, and *proximity arcs* make sure that a label will be placed close to its point-feature. Inverse to the proximity arcs are the *boundary arcs* which prevent that a label λ will be placed so that its point-feature $a(\lambda)$ lies in the interior of the corresponding rectangle. Finally, we introduce the *label separation arcs* which control mutual overlaps between possibly overlapping labels.

Modeling Point-Features

Again, let $\Pi = \{a(\lambda) \mid \lambda \in \Lambda\}$ be the set of point-features in an instance of a labeling problem $I = (\Lambda, w, h, a)$, and let $k = |\Pi|$ be their number. Clearly, the positions of the k point-features are specified in the input set Π. In a pair of labeling graphs $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ for I, let $x_i \in N_x$ and $y_i \in N_y$ be the nodes for each $p_i \in \Pi$. We fix the positions of the point-features by inserting four directed paths

$$P_x = (x_1, \ldots, x_k) \qquad P_{-x} = (x_k, \ldots, x_1)$$
$$P_y = (y_1, \ldots, y_k) \qquad P_{-y} = (y_k, \ldots, y_1)$$

with weights

$$w_{(x_i,x_{i+1})} = (p_{i+1})_x - (p_i)_x \qquad w_{(x_{i+1},x_i)} = (p_i)_x - (p_{i+1})_x$$
$$w_{(y_i,y_{i+1})} = (p_{i+1})_y - (p_i)_y \qquad w_{(y_{i+1},y_i)} = (p_i)_y - (p_{i+1})_y$$

for $i \in \{1, \ldots, k-1\}$. We call the directed edges on these paths *fixed distance arcs* and refer to them as A_F. Figure 5.4 shows a set of point-features and its representation in the constraint graphs.

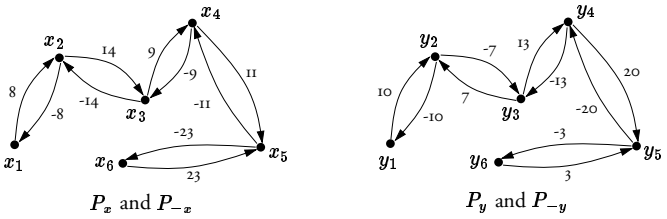


Figure 5.4: Modeling the placement of point-features with fixed distance arcs A_F

Lemma 5.1. *Let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be a pair of labeling graphs for an instance I of a labeling problem which contain the fixed distance arcs,* i.e., *$A_F \subseteq A_x \cup A_y$. A coordinate assignment c that respects A_F results in a correct placement of point-features (up to translation).*

Proof. We have $c(x_{i+1}) - c(x_i) \geq (p_{i+1})_x - (p_i)_x$ and $c(x_i) - c(x_{i+1}) \geq (p_i)_x - (p_{i+1})_x$, thus $c(x_{i+1}) - c(x_i) = (p_{i+1})_x - (p_i)_x$ for every $i \in \{1, \ldots, k-1\}$. The same holds for the y-coordinates. Since each node is linked to each other node by two inverse subpaths of each P_x and P_{-x} and P_y and P_{-y}, the fixed relative position of the point-features follows with Observation 3.2. ∎

Remark 5.1. As the proof of Lemma 5.1 shows, any arc set which corresponds to the equality constraints arising from the fixed distances of the point-features can be taken here, *e.g.*, also a simple directed cycle which links all point-features. We will discuss our choice in the implementation of our exact algorithms on page 145 in Section 5.4.1.

Modeling Labels

Each label $\lambda \in \Lambda$ has to be represented by a rectangle $R(\lambda)$ of width $w(\lambda)$ and height $h(\lambda)$. Additionally, we have to ensure that λ will be placed correctly with respect to its point-feature $a(\lambda)$, *i.e.*, $a(\lambda)$ must lie on the boundary of $r(\lambda)$ and must satisfy additional properties depending on the labeling model.

To model the sizes of the rectangle $R(\lambda)$ corresponding to the label λ we straightforwardly introduce four *label size arcs*

$$A_L(\lambda) = \{(l_\lambda, r_\lambda), (r_\lambda, l_\lambda), (b_\lambda, t_\lambda), (t_\lambda, b_\lambda)\}$$

for each label λ. The weights of these arcs are

$$w_{(l_\lambda,r_\lambda)} = w(\lambda) \qquad w_{(r_\lambda,l_\lambda)} = -w(\lambda)$$
$$w_{(b_\lambda,t_\lambda)} = h(\lambda) \qquad w_{(t_\lambda,b_\lambda)} = -h(\lambda) \ ,$$

see Figure 5.5.

Similar to the fixed distance arcs, the two pairs of label size arcs result in two equality constraints which fix the distances between the boundaries of a label.

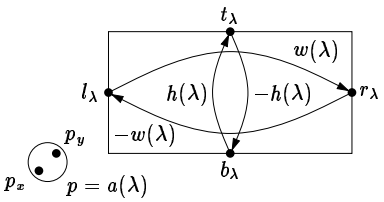


Figure 5.5: Modeling the size of a label λ with label size arcs $A_L(\lambda)$

In addition to coding the size constraints we have to make sure that the rectangle $R(\lambda)$ of a label λ is placed close to the appropriate point-feature $a(\lambda)$. Let p_x and p_y be the nodes representing point $a(\lambda)$ in the constraint graphs. We add four *proximity arcs*

$$A_P(\lambda) = \{(p_x, r_\lambda), (l_\lambda, p_x), (p_y, t_\lambda), (b_\lambda, p_y)\}$$

as illustrated in Figure 5.6. These arcs have zero weight and exclude placements of a label λ so that its point-feature $a(\lambda)$ lies outside the rectangle $R(\lambda)$. To see this, we write down the corresponding constraints for the associated assignments c_x and c_y:

$$c_x(r_\lambda) - c_x(p_x) \geq 0 \qquad c_x(l_\lambda) - c_x(p_x) \geq 0 \tag{5.1}$$
$$c_y(t_\lambda) - c_y(p_y) \geq 0 \qquad c_y(b_\lambda) - c_y(p_y) \geq 0 \ . \tag{5.2}$$

By rewriting the pairs of inequalities (5.1) and (5.2) as

$$c_x(l_\lambda) \leq c_x(p_x) \leq c_x(r_\lambda) \tag{5.3}$$
$$c_y(b_\lambda) \leq c_y(p_y) \leq c_y(b_\lambda) \tag{5.4}$$

we get a more apparent characterization of the placement restriction modeled by the proximity arcs.

Nevertheless, the point-feature $a(\lambda)$ may still lie inside the rectangle $R(\lambda)$ and may not be conform to the labeling model. Depending on the model, we define a constant

$$d = \begin{cases} 1 & \text{in slider models} \\ 2 & \text{in discrete models} \end{cases}$$

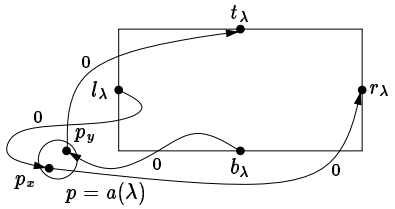


Figure 5.6: Modeling the proximity of a label λ and its point-feature $a(\lambda)$ with proximity arcs $A_P(\lambda)$

and disallow this kind of placement by requiring at least d of the maximally four *boundary arcs*[2], see also Figure 5.7.

$$A_B(\lambda) = \begin{cases} \{(r_\lambda, p_x), (p_x, l_\lambda), (t_\lambda, p_y), (p_y, b_\lambda)\} & \text{in the four-slider or four-pos. model} \\ \{(r_\lambda, p_x), (p_x, l_\lambda), (p_y, b_\lambda)\} & \text{in the two-position model} \\ \{(t_\lambda, p_y), (p_y, b_\lambda)\} & \text{in the two-slider model} \\ \{(p_x, l_\lambda), (p_y, b_\lambda)\} & \text{in the one-position model} \\ \{(p_y, b_\lambda)\} & \text{in the one-slider model} \end{cases}$$

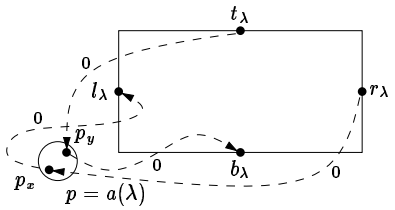


Figure 5.7: Determining the labeling model. Label λ will be placed according to the model due to at least d of the boundary arcs $A_B(\lambda)$

Note that the boundary arcs are inverse to the proximity arcs for label λ. If, *e.g.*, (r_λ, p_x) is present in D_x, it forces—together with its inverse proximity arc (p_x, r_λ)—the coordinate of the right side of $r(\lambda)$ to be equal to the coordinate of x; the label has to be placed at its leftmost position. Note the similarity to property (L2) in Definition 5.2 on page 116. Each arc in $\{(r_\lambda, p_x), (p_x, l_\lambda), (t_\lambda, p_y), (p_y, b_\lambda)\}$ corresponds to a boundary of λ. Clearly, in slider models only one of the arcs must be present; it can be chosen arbitrarily in the four-slider model, it must be a vertical arc in the two-slider model, and it must be (p_y, b_λ) in the one-slider model. In the discrete models we must choose two of four arcs in the four-position model and two of three in the two-position model. In the one-position model, the corner of the point-feature is fixed and characterized through the

[2] Allowing d to be zero leads to a different continuous model in which a label may also overlap its point-feature. In map labeling applications, however, this model is not relevant.

arcs (p_x, l_λ) and (p_y, b_λ). Note that we can express all six axis-parallel rectangular labeling models in Figure 1.3 on page 10 as additional requirements on the constraint graphs.

Having introduced the fixed distance arcs, the label size arcs and the boundary arcs, we can provide a combinatorial formulation of a *labeling* as defined in Section 5.1. Note that this definition ignores overlaps; according to Definition 5.2, a labeling is just a placement of labels which satisfies the appropriate labeling model.

Lemma 5.2. *Let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be a pair of labeling graphs for an instance $I = (\Lambda, w, h, a)$ of a labeling problem, and let $\Lambda_P \subseteq \Lambda$ be a subset of labels. Let A_x and A_y contain the fixed distance arcs A_F, and for each label in Λ_P the proximity arcs $A_P(\lambda)$ and at least d boundary arcs $B(\lambda) \in \binom{A_B(\lambda)}{d}$, i.e.,*

$$A_L = A_F \cup \bigcup_{\lambda \in \Lambda_P} A_P(\lambda) \cup \bigcup_{\lambda \in \Lambda_P} B(\lambda) \ .$$

Then a coordinate assignment c that respects A_L results in a labeling for $\Lambda_P \in \Lambda$, that is, in a placement in which each label is represented by a rectangle $R(\lambda)$ of width $w(\lambda)$ and height $h(\lambda)$. Each label is placed so that point-feature $a(\lambda)$ lies on the boundary of $R(\lambda)$ if $d = 1$ and on a corner of $r(\lambda)$ if $d = 2$ according to the appropriate model.

Proof. We construct a labeling ρ and show for each label $\lambda \in \Lambda_P$ that the two properties of a labeling hold, see Definition 5.2. The coordinate assignment c determines the lower left corner and upper right corner of rectangle $\rho(\lambda)$:

$$\begin{aligned} ll(\rho(\lambda)) &= \big(c(l_\lambda), c(b_\lambda)\big) \\ ur(\rho(\lambda)) &= \big(c(t_\lambda), c(r_\lambda)\big) \ . \end{aligned}$$

Let x and y be the coordinates of point-feature $a(\lambda)$ and let R be the rectangle $\rho(\lambda)$. Since c respects $A_L(\lambda)$, we have $ur(R)_x - ll(R)_x = w(\lambda)$ and $ur(R)_y - ll(R)_y = h(\lambda)$. Theorem 3.1 and the fact that the two assignments in c are feasible imply that $B(\lambda)$ contains at most one vertical and one horizontal arc—otherwise it would induce a positive cycle together with a proximity arc. We get $ur(R)_x - x \geq 0$, $x - ll(R)_x \geq 0$, $ur(R)_y - y \geq 0$ and $y - ll(R)_y \geq 0$, since c respects $A_P(\lambda)$. If $d = 1$, then at least one inequality becomes an equality, if $d = 2$ one horizontal and one vertical inequality become equalities. It follows that $a(\lambda)$ lies on a boundary of R and on a corner if $d = 2$. ∎

Controlling Overlaps

Lemma 5.2 guarantees that the placement satisfies the requirements of a labeling. In all labeling problems we have to control the number of overlaps between the labels—it must be zero in the label number and label size maximization problems, and small in the label overlap minimization problem. It is easy to see that it suffices to consider only the pairs of labels that can possibly interact. This set is in general much smaller than the potential $|\Lambda_P|(|\Lambda_P| - 1)$ overlaps.

Consider two different labels λ and μ and the corresponding rectangles $R = \rho(\lambda)$ and $Q = \rho(\mu)$. We call the pair *vertically separated* if R is placed either above or below

Q. Similarly, λ and μ are *horizontally separated* if one rectangle is placed to the left of the other. As for the segments in the chapter on compaction, a label pair is *separated* if it is either horizontally or vertically separated. Obviously, two labels overlap if they are not separated. In our combinatorial formulation, we model separation by the *label separation arcs*

$$A_S(\lambda, \mu) \in \mathcal{P}\big(\{(t_\mu, b_\lambda), (t_\lambda, b_\mu), (r_\mu, l_\lambda), (r_\lambda, l_\mu)\}\big) \ .$$

Label separation arcs have weight zero.

Similar as for the boundary arcs, which are determined by the labeling model, the separation arcs depend on the relative position of the two point-features which belong to the labels λ and μ. If these are far away, we do not need any separation arcs at all. Generally, the position of the regions in which the labels can be placed determines the set $A_S(\lambda, \mu)$.

Let $\mathcal{R}_\lambda$ be the boundary of the region in which label λ can be placed. Note that the size of $\mathcal{R}_\lambda$ depends on the labeling model. In the four-position or four-slider model the region is defined by lower left corner $(a(\lambda)_x - w(\lambda), a(\lambda)_y - h(\lambda))$ and upper right corner $(a(\lambda)_x + w(\lambda), a(\lambda)_y + h(\lambda))$. Likewise, we determine $\mathcal{R}_\mu$ for label μ. If the intersection of $\mathcal{R}_\lambda$ and $\mathcal{R}_\mu$ is empty, λ and μ can never overlap, and we do not have to add any label separation arcs for this pair. In this case we set $A_S(\lambda, \mu) = \varnothing$.

Consider now the case that the intersection of $\mathcal{R}_\lambda$ and $\mathcal{R}_\mu$ is not empty, as depicted in Figure 5.8. Depending on the position of the corresponding point-features $a(\lambda)$ and $a(\mu)$, the set $A_S(\lambda, \mu)$ contains the following label separation arcs:

1. If $a(\mu)_x \geq a(\lambda)_x$ we have $(r_\lambda, l_\mu) \in A_S(\lambda, \mu)$.
2. If $a(\lambda)_x \geq a(\mu)_x$ we have $(r_\mu, l_\lambda) \in A_S(\lambda, \mu)$.
3. If $a(\mu)_y \geq a(\lambda)_y$ we have $(t_\lambda, b_\mu) \in A_S(\lambda, \mu)$.
4. If $a(\lambda)_y \geq a(\mu)_y$ we have $(t_\mu, b_\lambda) \in A_S(\lambda, \mu)$.

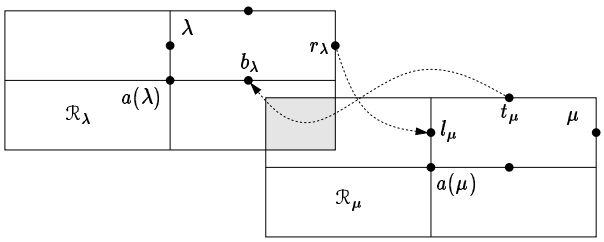


Figure 5.8: Label separation arcs between two labels λ and μ. In the example, the regions $\mathcal{R}_\lambda$ and $\mathcal{R}_\mu$ are built according to the four-position or four-slider model, and we have $a(\mu)_x \geq a(\lambda)_x$ and $a(\lambda)_y \geq a(\mu)_y$

Note that the only case in which $A_S(\lambda, \mu)$ contains all four possible label separation arcs occurs if λ and μ label the same point-feature, *i.e.*, $a(\lambda) = a(\mu)$.

Lemma 5.3. *Let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be a pair of labeling graphs for an instance $I = (\Lambda, w, h, a)$ of a labeling problem, let $\Lambda_P \subseteq \Lambda$ be a subset of labels, and let λ and μ be two different labels in Λ_P. Let A_x and A_y contain the arc set A_L as defined in Lemma 5.2.*

Let c be a coordinate assignment which respects the arcs in A_L. If the set $A_S(\lambda, \mu)$ is empty or c respects at least one of the arcs in $A_S(\lambda, \mu)$, then λ and μ do not overlap in the labeling.

Proof. By Lemma 5.2 we know that the placement is a labeling. We distinguish the following two cases:

Case 1: The set $A_S(\lambda, \mu)$ is empty.
By construction of the set of label separation arcs for λ and μ the pair cannot overlap in a labeling, since the regions $\mathcal{R}_\lambda$ and $\mathcal{R}_\mu$ are disjunct.

Case 2: The set $A_S(\lambda, \mu)$ is not empty.
In this case, the coordinate assignment respects one of the label separation arcs. Depending on the arc, λ is either left of, right of, below, or above μ in the corresponding labeling. ∎

5.2.2 Combinatorial Reformulations

At this point, we can reformulate the labeling problems in Section 5.1 as combinatorial problems in the labeling graphs. As for the compaction problems in the previous chapter, we model the space of feasible solutions for instances of the appropriate labeling problem in terms of properties of the corresponding labeling graphs. Like for compaction we start with fixed horizontal and vertical arc sets—in this case the fixed distance arcs, the label size arcs and the proximity arcs—for which we want to find an extension by adding additional arcs—in the labeling graphs these are the boundary arcs and separation arcs.

Following the notation of the preceding chapter, we refer to the boundary and label separation arcs as *potential arcs*

$$A_{\text{pot}} = \bigcup_{\lambda \in \Lambda} A_B(\lambda) \cup \bigcup_{\{\lambda,\mu\} \in \binom{\Lambda}{2}} A_S(\lambda, \mu) \ .$$

We will now present combinatorial equivalents of the different problems which will be the bases of ILP formulations in the next section. One of the main tasks will be to choose additional arcs from the set of potential arcs without creating positive directed cycles. In the following, we will assume $d = 1$ for the slider models and $d = 2$ for the discrete models, as defined on page 124. We begin with the pure labeling problem LAB in which all labels must be placed in their original size without any overlaps and state the corresponding combinatorial problem:

Definition 5.10 (Labeling Graph Satisfaction problem, LS). Let $I = (\Lambda, w, h, a)$ be an instance of a labeling problem, and let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be the pair of labeling graphs which contain the fixed distance arcs, the label size arcs and the proximity arcs. Find a subset of potential arcs $A \subseteq A_{\text{pot}}$ with the properties:

(S1) $|A \cap A_B(\lambda)| \geq d$ for all $\lambda \in \Lambda$.

(S2) $|A \cap A_S(\lambda, \mu)| \geq 1$ for all $\{\lambda, \mu\} \in \binom{\Lambda}{2}$, $\mathcal{R}_\lambda \cap \mathcal{R}_\mu \neq \varnothing$.

(S3) $A \cup A_x \cup A_y$ does not contain a positive cycle.

Remark 5.2. Properties (S2) and (S3) extend the notion of completeness as introduced for the placement graphs in Definition 4.7 on page 47.

Theorem 5.1. *Problems* LAB *and* LS *are polynomially equivalent.*

Proof. Let A be a solution of LS. We extend (D_x, D_y) by adding A to the arc sets. Because of (S3) and Theorem 3.1, a coordinate assignment exists that respects both the horizontal and vertical arc set. We place the labels as in the proof of Lemma 5.2 where we show that the placement is a labeling. By Lemma 5.3 we know that the number of overlaps in the labeling is zero, since for each pair of labels the set of label separation arcs is either empty or the coordinate assignment respects at least one of its arcs. These properties characterize a solution for the labeling problem LAB.

For the other direction, we start with the given coordinate assignment c resulting from the placement of labels. We create the set A of additional arcs as follows: For each label λ we add one or two boundary arcs, depending on how the rectangle $R(\lambda)$ is placed with respect to point-feature $a(\lambda)$. Similarly, we add appropriate arcs from $A(\lambda, \mu)$ for pairs of labels λ and μ, depending on the relative position of $R(\lambda)$ and $R(\mu)$ in the labeling. Note that we choose the additional arcs so that they are respected by c. Properties (S1) and (S2) follow by construction, property (S3) follows by Theorem 3.1. ∎

In the next section, we will propose an algorithm for the label size maximization problem which is based on repeatedly solving the combinatorial problem LS in a binary search manner. We exploit the observation in (Kučera *et al.*, 1993) that it is sufficient to solve a quadratic number of labeling problems to determine the optimal scale factor.

In the label number maximization problem, we have to compute a largest subset of labels for which a labeling without overlaps exists. The combinatorial equivalent is as follows:

Definition 5.11 (Maximum Labeling Graph Satisfaction problem, MLS).
Let $I = (\Lambda, w, h, a)$ be an instance of a labeling problem, and let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be the pair of labeling graphs which contain the fixed distance arcs, the label size arcs and the proximity arcs. Find a set of labels $\Lambda_P \subseteq \Lambda$ of greatest cardinality and a subset of potential arcs $A \subseteq A_{\text{pot}}$ with the properties:

(M1) $|A \cap A_B(\lambda)| \geq d$ for all $\lambda \in \Lambda_P$.

(M2) $|A \cap A_S(\lambda, \mu)| \geq 1$ for all $\{\lambda, \mu\} \in \binom{\Lambda_P}{2}$, $\mathcal{R}_\lambda \cap \mathcal{R}_\mu \neq \varnothing$.

(M3) $A \cup A_x \cup A_y$ does not contain a positive cycle.

Theorem 5.2. *Problems* LNM *and* MLS *are polynomially equivalent.*

Proof. Let Λ_P and A be a solution of the combinatorial problem. The arc set A corresponds to a feasible solution of the instance $(\Lambda_P, w|_{\Lambda_P}, h|_{\Lambda_P}, a|_{\Lambda_P})$ by Theorem 5.1.

For the backward direction let Λ_P be the set of placed labels. We construct the arc set as in the equivalence proof for the pure labeling problem. With the same argument as above this results in a solution of MLS, since the pair (Λ_P, A) satisfies properties (M1)-(M3). ∎

Definition 5.12 (Minimum Labeling Graph Overlap Satisfaction problem, MLOS). Let $I = (\Lambda, w, h, a)$ be an instance of a labeling problem, and let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be the pair of labeling graphs which contain the fixed distance arcs, the label size arcs, and the proximity arcs. Find a set of label pairs $L_{\text{sep}} \subseteq \binom{\Lambda}{2}$ of maximum size and a subset of potential arcs $A \subseteq A_{\text{pot}}$ with the properties:

(O1) $|A \cap A_B(\lambda)| \geq d$ for all $\lambda \in \Lambda$.

(O2) $|A \cap A_S(\lambda, \mu)| \geq 1$ for all $\{\lambda, \mu\} \in L_{\text{sep}}$.

(O3) $A \cup A_x \cup A_y$ does not contain a positive cycle.

Theorem 5.3. *Problems* LOM *and* MLOS *are polynomially equivalent.*

Proof. Let A and L_{sep} be a solution for the combinatorial problem. By Lemma 5.2 we construct a labeling for Λ. By Lemma 5.3 $|L_{\text{sep}}|$ pairs of labels are overlap-free.

For the other direction we construct L_{sep} and A from a given labeling with k pairs of overlapping labels. The construction of the arc set A is as in the proof of Theorem 5.1. For each pair of labels that do not overlap we add at least one appropriate label separation arc as in the equivalence proof for the pure labeling problem. Similar arguments as in the above proofs show that the resulting arc set satisfies properties (O1)-(O3). ∎

5.3 Integer Linear Programming Formulations

The previous section shows how to transform the labeling problems LAB, LNM, and LOM which ask for a labeling without overlaps, an overlap-free labeling for a maximal number of labels, and a labeling with the minimum number of overlaps, respectively, into the combinatorial problems LS, MLS, and MLOS involving the labeling graphs. We will now provide integer linear programs for the combinatorial equivalents and focus on the pure labeling problem, the label number maximization problem, and the label size maximization problem. The ILP formulations will enable us to devise exact LP-based algorithms for the labeling problems of Section 5.1, including the label number maximization problem in all models and the label size maximization problem in the discrete models.

We propose two different zero-one ILP formulations for the problems and show that the emerging separation problem for the class of positive cycle inequalities is unfortunately *NP*-hard. We present an extended formulation with additional continuous variables in which we can omit the positive cycle inequalities. The prize for evading the difficult separation problem is the additional variables and the formulation as a "big M approach".

We show that feasible solutions of the integer linear programs correspond to feasible labelings and vice versa. In particular, we will exploit this one-to-one correspondence for developing exact algorithms for the different labeling problems.

In Section 5.3.1, we describe the labeling polytope and introduce three non-trivial classes of inequalities for the pure labeling problem LAB. We use these inequalities to present a zero-one ILP for LAB in Section 5.3.2. We prove in Section 5.3.3 that the problem of finding violated positive cycle inequalities is *NP*-complete and provide an equivalent ILP formulation without these inequalities in Section 5.3.4. Finally, Section 5.3.5 demonstrates how to modify the formulations in order to optimize over the set of feasible solutions for the label number maximization problem.

5.3.1 The Labeling Polytope and the Labeling Problem

Similar to the compaction problems we characterize the space of feasible solutions of the pure labeling problem as a zero-one polytope. We want to describe extensions of the labeling graphs that can be obtained by adding a subset of the potential arcs so that certain properties are satisfied. Because of the similarity to the compaction problems, we use the term *extension* with the same meaning for labeling graphs—that is, for a pair of graphs that results from adding potential arcs to the original graphs. We refer by $\widehat{A}$ to the set of potential arcs which are part of the extension.

Let D_x and D_y be a pair of labeling graphs which contain only the fixed distance arcs A_F, the label size arcs $\cup_{\lambda\in\Lambda} A_S(\lambda)$, and the proximity arcs $\cup_{\lambda\in\Lambda} A_P(\lambda)$. By Theorem 5.1, the set of feasible solutions for the pure labeling problem is

$$\mathcal{L}(D_x, D_y) = \{\widehat{A} \in A_{\text{pot}} \mid \widehat{A} \text{ satisfies (S1), (S2), and (S3) in Definition 5.10}\}$$

Like in compaction, we use an incidence vector for the set of potential arcs to characterize the labeling polytope. Each set $\widehat{A} \in \mathcal{L}(D_x, D_y)$ defines an element $x^{\widehat{A}}$ of the vector space $\mathbb{Q}^{A_{\text{pot}}}$ with the interpretation

$$x_a^{\widehat{A}} = \begin{cases} 1 & a \in \widehat{A} \\ 0 & \text{otherwise} \end{cases} .$$

The incidence vectors enable us to characterize the *labeling polytope* for an instance $I = (\Lambda, w, h, a)$ of the pure labeling problem. Let (D_x, D_y) be the labeling graphs that only contain the fixed distance arcs, the label size arcs and the proximity arcs. The labeling polytope is then

$$P_{\text{LAB}} = \text{conv}\{x^{\widehat{A}} \in \mathbb{Q}^{A_{\text{pot}}} \mid \widehat{A} \in \mathcal{L}(D_x, D_y)\} .$$

We will show that the following four classes of inequalities together with integrality constraints for the solution vectors result in a description of the integral points in the labeling polytope.

– **Trivial inequalities.** As for the compaction problems, x is a characteristic vector:

$$0 \leq x_a \leq 1 \qquad \forall a \in A_{\text{pot}} \tag{5.5}$$

– **Integrality constraints.**. In addition, x must be integral:

$$x_a \in \{0,1\} \qquad \forall a \in A_{\text{pot}} \tag{5.6}$$

– **Boundary inequalities.** These inequalities ensure that the resulting labeling is in the correct model: Depending on the constant d, at least one or two of the boundary arcs have to be present in the extension. Note that the inequality can be strengthened to an equality (we will exploit this in one of the forthcoming ILP formulations).

$$\sum_{a \in A_B(\lambda)} x_a \geq d \qquad \forall \lambda \in \Lambda \tag{5.7}$$

– **Label separation inequalities.** This class of inequalities prevents overlaps between pairs of labels. At least one of the arcs of the non-empty sets of label separation arcs must be present in a solution of LS, the combinatorial equivalent of the labeling problem.

$$\sum_{a \in A_S(\lambda,\mu)} x_a \geq 1 \qquad \forall \{\lambda,\mu\} \in \binom{\Lambda}{2}, \mathcal{R}_\lambda \cap \mathcal{R}_\mu \neq \varnothing \tag{5.8}$$

– **Positive cycle inequalities.** A considerable advantage of our ILP formulations for labeling problems over the formulations for compaction problems is that we do not need an additional coordinate vector and can concentrate on a pure zero-one polytope. We have to exclude positive cycles, however, in order to guarantee the feasibility of the underlying assignments. We explicitly prevent positive cycles by a potentially exponential number of positive cycle inequalities:

$$\sum_{a \in C \cap A_{\text{pot}}} x_a \leq |C \cap A_{\text{pot}}| - 1 \qquad \forall \text{ positive cycles } C \text{ in extension} \tag{5.9}$$

5.3.2 Zero-One Formulation for LAB

Let $I = (\Lambda, w, h, a)$ be an instance of the pure labeling problem, and let D_x and D_y be the corresponding labeling graphs with fixed distance arcs, label size arcs, and proximity arcs. Again, the constant d is equal to one for the slider models and equal to two for the discrete models.

We use inequalities (5.5)-(5.9) to formulate the integer linear program for the labeling problem LAB. Since Theorem 5.4 will show that inequalities (5.5)-(5.9) indeed characterize the integral points within the labeling polytope, and the task in the labeling problem is to find just one feasible solution, we can use an arbitrary objective function. Let $z \in \mathbb{Q}^{A_{\text{pot}}}$ be an arbitrary coefficient vector; we will propose a convenient choice for z in a following remark. The integer linear program for the labeling problems is as follows:

$$\begin{aligned}
\max \quad & z^T x && && \text{(ILP.4)}\\
\text{subject to} \quad & \sum_{a \in A_B(\lambda)} x_a \geq d && \forall \lambda \in \Lambda && \text{(ILP.4.1)}\\
& \sum_{a \in A_S(\lambda,\mu)} x_a \geq 1 && \forall \{\lambda, \mu\} \in \binom{\Lambda}{2}, \mathcal{R}_\lambda \cap \mathcal{R}_\mu \neq \varnothing && \text{(ILP.4.2)}\\
& \sum_{a \in C \cap A_{\text{pot}}} x_a \leq |C \cap A_{\text{pot}}| - 1 && \forall \text{ positive cycles } C && \text{(ILP.4.3)}\\
& x_a \in \{0, 1\} && \forall a \in A_{\text{pot}} && \text{(ILP.4.4)}
\end{aligned}$$

Theorem 5.4. *Let $I = (\Lambda, w, h, a)$ be an instance of the labeling problem* LAB*. A feasible solution x of (*ILP.4*) for the combinatorial version of* LAB *corresponds to a feasible solution of* LAB *and vice versa.*

Proof. A feasible solution of (ILP.4) is an incidence vector for the set of potential additional arcs. The inequalities model exactly the requirements (S1)-(S3) in the combinatorial equivalent of LAB. By Theorem 5.1 we know that the solutions of the combinatorial problem are in one-to-one correspondence with the solutions of the pure labeling problem. ∎

Remark 5.3. As we will show in the next section, the most difficult part in the above integer linear programming formulation is to find violated positive cycle inequalities. We therefore propose to minimize the number of additional arcs in order to create as few cycles as possible. This corresponds to $z = -\mathbb{1}$ in (ILP.4).

5.3.3 Complexity of Finding Positive Cycles

Unfortunately, the number of potential positive cycles in the extension is exponential, distinguishing the class of positive cycle inequalities from the other classes. We have a linear number of trivial and boundary inequalities and a possibly quadratic number of label separation inequalities. A crucial issue in solving our integer linear programming formulations for the labeling problems concerns finding violated positive cycle inequalities in an LP solution of the relaxed ILP. The decision version of the corresponding separation problem is the following problem:[3]

Definition 5.13 (Separation problem for positive cycle inequalities, decision version). Given a digraph $D = (N, A)$, a weight vector $d \in \mathbb{Z}^A$ and an LP-solution $\chi \in \mathbb{Q}^A$, $0 \leq \chi_a \leq 1 \; \forall a \in A$. Does a directed cycle $C \subseteq A$ with

$$\begin{aligned}
\sum_{a \in C} \chi_a &> |C| - 1 \quad \text{and}\\
\sum_{a \in C} d_a &> 0 \quad \text{exist?}
\end{aligned}$$

[3] For the sake of simplicity, we assume that additional variables $\chi_a = 1$ for each fixed arc $a \in A$ exist and that all arc weights are integral.

We proceed as in the easier case with non-negative arc weights (see Definition 4.14 on page 91) and substitute $\xi_a = 1 - \chi_a$ for all $a \in A$. The substitution results in the following equivalent version of the above separation problem.

Definition 5.14 (Positive Weight-Constrained Cycle problem, PWCC). Given a digraph $D = (V, A)$, a weight vector $d \in \mathbb{Z}^A$, and a vector $\xi \in \mathbb{Q}^A$, $0 \leq \xi_a \leq 1$ $\forall a \in A$. Does a directed cycle $C \subseteq A$ with

$$\begin{aligned} \sum_{a \in C} \xi_a &< 1 \quad \text{and} \\ \sum_{a \in C} d_a &> 0 \quad \text{exist?} \end{aligned}$$

The following problem is *NP*-complete (see Garey and Johnson, 1979, [ND30]):

Definition 5.15 (Shortest Weight-Constrained Directed Path problem, SWCP). Given a directed graph $D = (N, A)$, a length function $l : A \to \mathbb{N}$, a weight function $w : A \to \mathbb{N}$, two specified vertices $s, t \in N$, and integers W and K. Does a simple directed path in D from s to t with total weight W or less and total length K or less exist?

We prove that the decision variant of the separation problem for positive cycle inequalities is *NP*-complete by reducing the positive weight-constrained cycle problem to the shortest weight-constrained directed path problem.

Theorem 5.5. *The positive weight-constrained cycle problem is NP-complete.*

Proof. The problem is in *NP* since a nondeterministic algorithm needs only guess a directed cycle C in D and check in polynomial time whether its weight is positive and whether $\xi(C) < 1$. Let $I_{\text{SWCP}} = \big((N, A),\, l,\, w,\, s,\, t,\, W,\, K\big)$ be an instance for the shortest weight-constrained directed path problem. We transform it in polynomial time into an instance $I_{\text{PWCC}} = (D', d, \xi)$ of the positive weight-constrained cycle problem.

Let $L = \max\{l(a) \mid a \in A\}$ be the length of the longest arc in A. We set $D' = (N, A')$ with $A' = A \cup \{(t, s)\}$ and define the two vectors as follows:

$$d_a = \begin{cases} -w(a) & a \in A \\ W + 1 & a = (t, s) \end{cases} \qquad \xi_a = \begin{cases} \frac{l(a)}{L+K+1} & a \in A \\ 1 - \frac{K+1}{L+K+1} & a = (t, s) \end{cases}.$$

Obviously, we have $d \in \mathbb{Z}^{A'}$ and $0 \leq \xi_a \leq 1$ for $a \in A'$.

We claim that there is a simple path from s to t in D with length at most K and weight at most W if and only if there is a directed cycle $C \subseteq A'$ with $\sum_{a \in C} d_a > 0$ and $\xi(C) < 1$.

Let P be such a path in D, and let C be the cycle which results from appending the arc (t, s) to P. Cycle C in D' has weight

$$\begin{aligned} \sum_{a \in C} d_a &= \sum_{a \in P} -w(a) + d_{(t,s)} = -\sum_{a \in P} w(a) + W + 1 \\ &\geq -W + W + 1 = 1 \\ &> 0 \end{aligned}$$

and ξ-value

$$\begin{aligned}\sum_{a\in C}\xi_a &= \sum_{a\in P}\left(\frac{l(a)}{L+K+1}\right)+1-\frac{K+1}{L+K+1}\\ &= \frac{1}{L+K+1}\underbrace{\sum_{a\in P}l(a)}_{\leq K}+\frac{L}{L+K+1}\\ &\leq \frac{K}{L+K+1}+\frac{L}{L+K+1}=\frac{L+K}{L+K+1}\\ &< 1\ .\end{aligned}$$

For the backward direction of the proof let C be a cycle in D' with positive weight and $\xi(C) < 1$. Note that the arc (t, s) must lie on C, since it is the only arc with positive weight d. It follows that the remaining arcs of C form a path P from s to t in D. The w-weight of P is

$$\begin{aligned}\sum_{a\in P}w(a) &= \sum_{a\in P}-d_a = W+1-\sum_{a\in P}d_a - W - 1\\ &= W+1-\underbrace{\sum_{a\in C}d_a}_{\geq 1}\\ &\leq W\ .\end{aligned}$$

The length of P is

$$\begin{aligned}\sum_{a\in P}l(a) &= \sum_{a\in P}(L+K+1)\xi_a\\ &= (L+K+1)\sum_{a\in P}\xi_a+(L+K+1)(1-\frac{K+1}{L+K+1})\\ &\quad -(L+K+1)(1-\frac{K+1}{L+K+1})\\ &= (L+K+1)\underbrace{\sum_{a\in C}\xi_a}_{<1}-(L+K+1)+K+1\\ &< L+K+1-(L+K+1)+K+1\\ &= K+1\ .\end{aligned}$$

Since all lengths are integral, we have $l(P) \leq K$. ∎

5.3.4 Extended Formulation

We introduce an alternative ILP formulation without positive cycle inequalities which is similar to the formulations in the chapter on compaction. In this *extended formulation* we

introduce additional continuous variables corresponding to the coordinates of the nodes in the constraint graphs. Let $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ be the labeling graphs like in the construction of the zero-one ILP in Section 5.3.2. As in the previous chapter, we call these variables the *coordinate vector* $c \in \mathbb{Q}^{N_x \cup N_y}$. We keep the boundary and label separation inequalities from the zero-one formulation, but we replace the positive cycle inequalities by two classes of inequalities which link the zero-one variables for the potential arcs x with the continuous variables c for the coordinates. Again, we can choose an arbitrary coefficient vector in the objective function.

– **Distance inequalities.** As for compaction, the coordinate vector represents the two assignments for the underlying constraint graphs. The class of distance inequalities corresponds to the shape inequalities and ensures the feasibility of the assignments for the fixed arcs in the labeling graphs.

$$c_j - c_j \geq w_a \qquad \forall a = (i,j) \in A_x \cup A_y \tag{5.10}$$

– **Consistency inequalities.** The class of consistency inequalities performs the same task as in the formulation for the compaction problems: Depending on their x-value, they activate or deactivate the constraints which correspond to potential arcs. The consistency inequalities form the link between the zero-one vector x and the coordinate vector c. As for compaction, we will show that this class of constraints indeed prohibits positive cycles in the extensions of the labeling graphs. Again, we use a "big M approach" and discuss the choice of the individual constants M_a in a following lemma. As in the previous chapter we must choose M_a as an upper bound on the distance $c_i - c_j$. In the case of map labeling problems we can determine tighter bounds as for the compaction problems.

$$c_j - c_i - M_a x_a \geq -M_a \qquad \forall a = (i,j) \in A_{\text{pot}} \tag{5.11}$$

Note that we omit the weights w_x and w_y: Potential arcs have zero weight in the labeling problems.

Lemma 5.4. *The following values are upper bounds on the distance $c_i - c_j$ and thus sufficient choices for M_a in the consistency inequality for potential arc a:*

$$M_a = \begin{cases} w(\lambda) & a = (i,j) \in A_B(\lambda), \{i,j\} \subseteq N_x \\ h(\lambda) & a = (i,j) \in A_B(\lambda), \{i,j\} \subseteq N_y \\ w(\lambda) + w(\mu) - (a(\mu)_x - a(\lambda)_x) & a = (i,j) \in A_S(\lambda,\mu), \{i,j\} \subseteq N_x \\ h(\lambda) + h(\mu) - (a(\mu)_y - a(\lambda)_y) & a = (i,j) \in A_S(\lambda,\mu), \{i,j\} \subseteq N_x \end{cases}.$$

Proof. We distinguish between the types of arcs:

Case 1: Arc a is a boundary arc.
Assume without loss of generality that $a \in A_B(\lambda)$ is a horizontal arc. One endnode of a corresponds to the left or right label boundary, the other one to the x-coordinate of the point-feature $a(\lambda)$. Due to the existence of the proximity arcs, the horizontal distance between a point-feature and the horizontal label boundaries is bounded by the width of the label.

Case 2: Arc a is a separation arc.
Again, we assume without loss of generality that a is horizontal, and, furthermore, that a is as in Figure 5.9; the other cases are symmetric. In the figure we illustrate the worst case: μ is at its leftmost and λ at its rightmost position. ∎

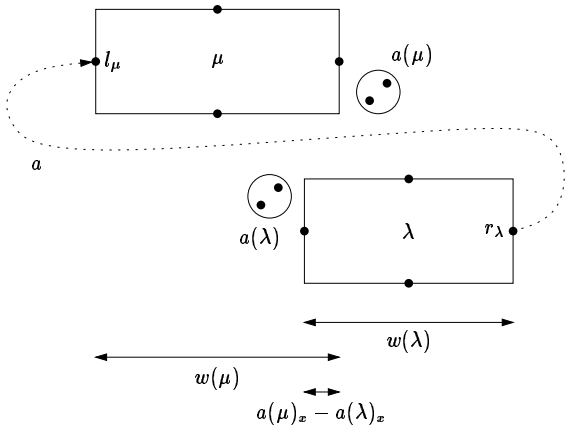


Figure 5.9: Label separation arc in the proof of Lemma 5.4

The extended formulation is:

$$\max \quad z^T \begin{pmatrix} x \\ c \end{pmatrix} \tag{EILP.1}$$

$$\text{subject to} \quad \sum_{a \in A_B(\lambda)} x_a \geq d \qquad \forall \lambda \in \Lambda \tag{EILP.1.1}$$

$$\sum_{a \in A_S(\lambda,\mu)} x_a \geq 1 \qquad \forall \{\lambda, \mu\} \in \binom{\Lambda}{2}, \mathcal{R}_\lambda \cap \mathcal{R}_\mu \neq \varnothing \tag{EILP.1.2}$$

$$c_j - c_i \geq w_a \qquad \forall a = (i,j) \in A_x \cup A_y \tag{EILP.1.3}$$

$$c_j - c_i - M_a x_a \geq -M_a \qquad \forall a = (i,j) \in A_{\text{pot}} \tag{EILP.1.4}$$

$$y_\lambda \in \{0,1\} \qquad \forall \lambda \in \Lambda \tag{EILP.1.5}$$

$$x_a \in \{0,1\} \qquad \forall a \in A_{\text{pot}} \tag{EILP.1.6}$$

As in the preceding chapter, the big M approach enables us to drop the positive cycle inequalities:

Lemma 5.5. *Let (x,c) be a feasible solution of (EILP.1), and let $(\widehat{D}_x, \widehat{D}_y)$ be the extensions corresponding to x. Then $\widehat{D}_x$ and $\widehat{D}_y$ do not contain positive cycles.*

Proof. The proof is like the proof of Lemma 4.16 on page 84 in Chapter 4. ∎

Theorem 5.6. *Let $I = (\Lambda, w, h, a)$ be an instance of the labeling problem* LAB*. A feasible solution (x, c) of (*EILP.1*) for the combinatorial version of* LAB *corresponds to a feasible solution of* LAB *and vice versa.*

Proof. The proof exploits Lemma 5.5 and is similar to the proof of Theorem 4.6 for the related formulation in the chapter on compaction. ∎

Although we manage to avoid positive cycle inequalities in the extended formulation we do this at the price of introducing more variables and constants M_a in order to link the decision variables with the continuous variables.

In Chapter 6 we exploit the similarity of the extended formulation to the integer linear programming formulations in the chapter on compaction in order to provide an algorithm for a combined compaction and labeling problem.

5.3.5 Label Number Maximization

The label number maximization problem differs from the pure labeling problem in the set of labels for which a labeling has to be found. Whereas an algorithm for the first problem must place all the labels or output that this is not possible, a feasible solution of the label number maximization problem LNM always exists, *e.g.*, by placing none of the labels. In order to develop a formulation for LNM, we must model the choice of the subset Λ_P that contains those labels from Λ that are placed. This choice affects the inequalities: We only have to ensure the correct labeling model and separation from other labels for the labels contained in Λ_P.

We present a first ILP formulation for the label number maximization problem which uses an additional incidence vector y to model the choice of Λ_P. The formulation extends the zero-one formulation for the pure labeling problem (ILP.4) by adapting the classes of inequalities so that they are only valid for labels in Λ_P. Later, we show how to modify this first formulation by a substitution of the newly introduced variables. The result is a zero-one integer linear program for LNM in which the only variables are the potential arc variables.

The additional incidence vector $y \in \mathbb{Q}^\Lambda$ has the interpretation

$$y_\lambda = \begin{cases} 1 & \lambda \in \Lambda_P \\ 0 & \lambda \in \Lambda \setminus \Lambda_P \end{cases} .$$

Our first formulation for LNM looks as follows:

$$\max \quad \sum_{\lambda \in \Lambda} y_\lambda \tag{ILP.5}$$

$$\text{subject to} \quad \sum_{a \in A_B(\lambda)} x_a \geq d \cdot y_\lambda \qquad \forall \lambda \in \Lambda \tag{ILP.5.1}$$

$$\sum_{a \in A_S(\lambda,\mu)} x_a - y_\lambda - y_\mu \geq -1 \quad \forall \{\lambda, \mu\} \in \binom{\Lambda}{2}, \mathcal{R}_\lambda \cap \mathcal{R}_\mu \neq \varnothing \tag{ILP.5.2}$$

$$\sum_{a \in C \cap A_{\text{pot}}} x_a \leq |C \cap A_{\text{pot}}| - 1 \qquad \forall \text{ positive cycles } C \tag{ILP.5.3}$$

$$y_\lambda \in \{0, 1\} \qquad \forall \lambda \in \Lambda \tag{ILP.5.4}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A_{\text{pot}} \tag{ILP.5.5}$$

Theorem 5.7. *Let $I = (\Lambda, w, h, a)$ be an instance of the label number maximization problem* LNM. *A feasible solution (y, x) of (*ILP.5*) for* MLS*, the combinatorial version of* LNM*, corresponds to a feasible solution of the label number maximization problem and vice versa. The value of the objective function is equal to the cardinality of Λ_P.*

Proof. The proof is similar as for the pure labeling problem: Each entry in (y, x) is zero or one because of the integrality conditions. We set $\Lambda_P = \{\lambda \in \Lambda \mid y_\lambda = 1\}$ and $A = \{a \in A_{\text{pot}} \mid x_a = 1\}$. Clearly, $|\Lambda_P| = \sum_{\lambda \in \Lambda} y_\lambda$. For each $\lambda \in \Lambda_P$, the boundary inequality turns to $\sum_{a \in A_B(\lambda)} x_a \geq d$, it follows (M1). Likewise, the appropriate separation inequality yields $\sum_{a \in A_S(\lambda,\mu)} x_a \geq 1$ if both λ and μ are in Λ_P, satisfying (M2). Obviously, the positive cycle inequalities ensure property (M3).

For the other direction, we set y_λ to one if $\lambda \in \Lambda_P$, and to zero otherwise. In the same manner, we set x_a to one if $a \in A$, otherwise to zero. Similar arguments as above show that (y, x) does not violate any of the inequalities, thus it is a feasible solution of (ILP.5). ∎

Corollary 5.1. *An optimal solution of (*ILP.5*) corresponds to an optimal solution of the label number maximization problem.*

At this point we use the observation that the boundary inequalities (5.7), introduced on page 132, can be changed to *boundary equalities*

$$\sum_{a \in A_B(\lambda)} x_a = d \qquad \forall \lambda \in \Lambda \tag{5.7b}$$

without affecting the proof of Theorem 5.7. In (ILP.5) for the label number maximization problem, we can make the same change and replace (ILP.5.1) by

$$\sum_{a \in A_B(\lambda)} x_a = d \cdot y_\lambda \qquad \forall \lambda \in \Lambda \ . \tag{ILP.5.1b}$$

This enables us to substitute

$$y_\lambda \quad \text{by} \quad \sum_{a \in A_B(\lambda)} \frac{x_a}{d} \qquad \text{for all } \lambda \in \Lambda \ . \tag{5.12}$$

Nevertheless, we have to be careful: the new formulation transfers the decision whether a label is placed or not to the boundary arcs. Two problems concerning the objective function occur:

– In the slider models, more than one boundary arc may be included in the extensions; placing a label according to a discrete model—which is then also feasible in a slider model—counts twice as much in the objective function. We therefore bound the number of inequalities depending on the labeling model and require

$$\sum_{a \in A_B(\lambda)} x_a \leq d \qquad \forall \lambda \in \Lambda \ . \tag{5.13}$$

In a discrete model, we can drop this class of inequalities. In an extension, at most two boundary arcs may be present, otherwise they induce a cycle of positive weight and the corresponding values of x violate the positive cycle inequalities. See also the discussion on *local cycles* on page 143.

– In the discrete models, we must exclude that only one boundary arc is present. Consider a label $\lambda \in \Lambda$ and its limits l_λ, r_λ, b_λ, and t_λ in the constraint graphs. Again, let p_x and p_y be the nodes which represent the point-feature $p = a(\lambda)$ in the labeling graphs. For each label λ we add the following equality constraint which ensures that the number of horizontal boundary arcs is equal to the number of vertical boundary arcs. For simplicity, assume that the variable x_a is equal to zero if a does not belong to the boundary arcs.

$$x_{(r_\lambda, p_x)} + x_{(p_x, l_\lambda)} = x_{(t_y, p_y)} + x_{(p_y, b_y)} \qquad \forall \lambda \in \Lambda \ . \tag{5.14}$$

Remark 5.4. Equalities (5.14) are according to the four-position model. Since $(t_\lambda, p_y) \notin A_B(\lambda)$ in the two-position model, (5.14) changes to

$$x_{(r_\lambda, p_x)} + x_{(p_x, l_\lambda)} = x_{(p_y, b_y)} \qquad \forall \lambda \in \Lambda \ . \tag{5.14b}$$

For similar reasons, we have

$$x_{(p_x, l_\lambda)} = x_{(p_y, b_y)} \qquad \forall \lambda \in \Lambda \ . \tag{5.14c}$$

in the one-position model.

The new ILP formulation is:

$$\max \quad \sum_{\lambda \in \Lambda} \sum_{a \in A_B(\lambda)} \frac{x_a}{d} \tag{ILP.6}$$

$$\text{subject to} \quad \sum_{a \in A_S(\lambda,\mu)} x_a - \sum_{a \in A_B(\lambda)} \frac{x_a}{d} - \sum_{a \in A_B(\mu)} \frac{x_a}{d} \geq -1$$

$$\forall \{\lambda, \mu\} \in \binom{\Lambda}{2}, \mathcal{R}_\lambda \cap \mathcal{R}_\mu \neq \varnothing \tag{ILP.6.1}$$

$$\sum_{a \in A_B(\lambda)} x_a \leq d \qquad \forall \lambda \in \Lambda \tag{ILP.6.2}$$

$$\sum_{a \in C \cap A_{\text{pot}}} x_a \leq |C \cap A_{\text{pot}}| - 1 \qquad \forall \text{ positive cycles } C \tag{ILP.6.3}$$

$$x_{(r_\lambda, p_x)} + x_{(p_x, l_\lambda)} = x_{(t_y, p_y)} + x_{(p_y, b_y)} \qquad \forall \lambda \in \Lambda \text{ if } d = 2 \tag{ILP.5.4}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A_{\text{pot}} \tag{ILP.6.4}$$

Theorem 5.8. *Let $I = (\Lambda, w, h, a)$ be an instance of the label number maximization problem* LNM. *A feasible solution x of* (ILP.6) *for* MLS, *the combinatorial version of* LNM, *corresponds to a feasible solution of the label number maximization problem and vice versa. The value of the objective function is equal to the cardinality of Λ_P.*

Proof. We show that a solution of (ILP.6) corresponds to a solution of (ILP.5), and that both objective functions take the same value.

Let x be a solution of (ILP.6). We define y_λ as in the substitution (5.12) and verify that (x, y) violates none of the inequalities of (ILP.5). Due to the additional inequalities (ILP.6.2) and (ILP.5.4), the values of the objective functions are equal. The positive cycle inequalities and the integrality constraints for the potential arc variables remain the same. Inequalities (ILP.5.1) and (ILP.5.2) hold because of the substitution.

Now consider a solution (x, y) of (ILP.5). The observation that the boundary inequalities can be changed to equalities without affecting the space of feasible solutions shows that x is also a solution of (ILP.6). ∎

Corollary 5.2. *An optimal solution of* (ILP.6) *corresponds to an optimal solution of the label number maximization problem.*

Remark 5.5. We can modify the zero-one formulations for the label number maximization problem in the same way as described in Section 5.3.4 in order to evade the positive cycle inequalities. Since the modification only consists of adding an additional coordinate vector and replacing the positive cycle inequalities by the distance and consistency inequalities, it does not affect the changes that enable us to optimize over the set of feasible solutions for the label number maximization problem.

5.4 Exact Labeling Algorithms

In this section, we develop exact algorithms for the map labeling problems that are based on integer linear programming formulations for the combinatorial problem equivalents.

We present different algorithms for both the zero-one and the extended formulation: these are branch-and-bound and branch-and-cut algorithms and an iterative branch-and-bound approach. The basic structure of the algorithms for the different labeling problems is very similar. We explain the algorithms for the label number maximization problem and demonstrate how to adapt the techniques to solve the pure labeling problem and the label size maximization problem.

The idea of the algorithms is based on the equivalence of the integer linear programming formulations and the original labeling problems in Section 5.1. Theorems 5.4-5.8 as well as Corollaries 5.1 and 5.2 and the respective proofs already suggest an algorithmic idea for attacking practical instances of the labeling problems: A solution of the integer linear programming formulations tells us which boundary and label separation arcs should be added to the arc sets of the labeling graphs (D_x, D_y). We use this information in a second step for computing the corresponding coordinate assignment with two separate feasible assignments, *e.g.*, with Algorithm 3.1.

We now provide a detailed description of the algorithms for the label number maximization problem:

A common pre-processing phase partitions a given instance of the problem in several components. Let G_Λ be the graph which contains a vertex for each label and an edge for each pair of labels λ and μ with non-empty intersection $R_\lambda \cap R_\mu$, *i.e.*, for pairs of labels which possibly overlap. It is easy to see that we can process the connected components of G_Λ separately.

A second phase of all exact algorithms deals with constructing the problem-dependent pair of constraint graphs for each component of the labeling problem. In the worst case, this takes quadratic time, which is due to the possibly quadratic number of label pairs that can overlap each other. This step consists of building the labeling graphs that contain the nodes for coordinates of point-features and label boundaries as well as the fixed distance, label size, and proximity arcs. In addition, the algorithm generates the set of potential additional arcs that contain the model-dependent boundary arcs and the label separation arcs.

The initial zero-one integer linear program for the label number maximization problem contains the boundary and separation inequalities. Due to the possibly large number of positive cycle inequalities, we only add a set of *local positive cycle inequalities*. We are able to determine these inequalities in advance by looking at positive cycles involving up to two labels:

Figure 5.10 illustrates the local cycles in the vertical case: The first type of cycles consists of two boundary arcs and a label size arc—in Figure 5.10, these are $\big((y_i, b_\lambda), (b_\lambda, t_\lambda), (t_\lambda, y_i)\big)$, and $\big((y_j, b_\mu), (b_\mu, t_\mu), (t_\mu, y_j)\big)$. We exclude these cycles by adding the two inequalities

$$\begin{aligned} x_{(y_i,b_\lambda)} + x_{(t_\lambda,y_i)} &\leq 1 \\ x_{(y_j,b_\mu)} + x_{(t_\mu,y_j)} &\leq 1 \ . \end{aligned}$$

Depending on the arc weights, a second type of positive cycle may appear which involves two labels linked by a label separation arc. If, like in Figure 5.10, the height of λ is greater

than the vertical distance between p_i and p_j, the cycle $\big((b_\lambda, t_\lambda), (t_\lambda, b_\mu), (b_\mu, y_j), (y_j, y_i), (y_i, b_\lambda)\big)$ has positive weight. In this case, we add the inequality

$$x_{(t_\lambda, b_\mu)} + x_{(y_i, b_\lambda)} \leq 1 \ .$$

A similar situation, $h(\mu) > (p_j)_y - (p_i)_y$, results in the inequality

$$x_{(t_\lambda, b_\mu)} + x_{(t_\mu, y_j)} \leq 1 \ .$$

If the sum of both heights exceeds the vertical distance between the point-features, *i.e.*, $h(\lambda) + h(\mu) > (p_j)_y - (p_i)_y$, we add the inequality

$$x_{(t_\lambda, b_\mu)} + x_{(t_\mu, y_j)} + x_{(y_i, b_\lambda)} \leq 2 \ .$$

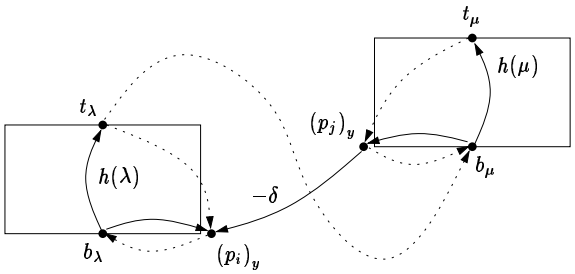


Figure 5.10: Local positive cycles in the vertical labeling graph D_y arising from the interaction of labels. Potential arcs are dashed, $\delta = (p_j)_y - (p_i)_y$

Our branch-and-cut algorithm combines a standard branch-and-bound scheme for solving the integer linear program with the following heuristic cutting plane algorithm: In order to find violated positive cycle inequalities we first disregard the weight of the cycles and identify violated cycle inequalities with Algorithm 4.6 on page 91. We add them to our formulation if the corresponding cycles have positive weight. A second strategy temporarily adds the potential arcs whose corresponding LP-value in the relaxation is above a certain threshold and finds positive cycles with Algorithm 3.1. We follow the advice of Cherkassky and Goldberg (1999) and use a fast version of the Bellman-Ford algorithm (subtree disassembly with update) in our heuristics. For every positive cycle we check if the corresponding LP-values constitute a violated inequality. If this is the case, we add the inequality to our ILP formulation.

Due to the hardness of the separation problem, finding violated positive cycle inequalities is quite difficult and often, the two heuristics fail. Therefore, we propose a second strategy that rests upon the fact that we can provide exact algorithms for the separation problem in the case that every component of the variable vector for the potential arcs is either equal to zero or to one. In that case, setting the threshold to one in the second of the above mentioned separation strategies always detects existing violated positive cycle inequalities.

We start with the initial integer linear programming formulation and proceed with the following iterative scheme, see also Algorithm 5.1. The core of our algorithm is a classical branch-and-bound framework for the current ILP. However, the aim in the first iterations is not to find optimal, but feasible solutions of the ILP. As discussed above, we can solve the separation problem in the case of integral LP-solutions exactly and add the resulting inequalities to the current ILP. If no violated inequalities exist, we double the value of k and iterate. When k reaches a threshold—experimental evaluation shows that 16 is a good choice for most problems—we do not interrupt the branch-and-bound algorithm and compute an optimal solution for the current ILP. In case the following step (line 6 of Algorithm 5.1) does not find a violated inequality, we know that the current solution is optimal for the entire ILP formulation for the label number maximization problem and that it results in an overlap-free labeling for a largest subset of labels.

Algorithm 5.1 *Iterative branch-and-bound scheme for the label number maximization problem*

Input: Initial ILP for the label number maximization problem LNM

Output: Optimal solution of entire ILP for LNM

1: current ILP = initial ILP;
2: $k = 1$;
3: **repeat**
4: **if** $k > 16$ **then** $k = \infty$;
5: solve current ILP with branch-and-bound, stop when kth feasible solution found;
6: $S =$ separated positive cycle inequalities;
7: **if** $S \equiv \varnothing$ **then**
8: $k = 2k$;
9: **else**
10: add S to current ILP;
11: **until** $S \equiv \varnothing$ and $k \equiv \infty$

The branch-and-bound scheme for the pure labeling problem is an easy special case of Algorithm 5.1. Note that the objective function is arbitrary and the optimization task consists of finding just a feasible solution. We can omit the parameter k and interrupt the optimization as soon as the branch-and-bound algorithm finds the first feasible solution.

In order to solve the label size maximization problem in the discrete models, we can use Algorithm 5.2 in a binary search manner. As illustrated in Section 5.1, the number of different scale factors that can lead to optimal solutions is at maximum quadratic, and we are able to identify these factors in advance. Performing a binary search over the set of candidate factors results in an exact algorithm for the label number maximization problem in the discrete models. The algorithm identifies the largest of these factors for which the pure labeling problem is not infeasible and provides an appropriate labeling.

Our algorithms for the extended formulation are similar to the branch-and-bound and branch-and-cut algorithms for the compaction problems. The advantage of the extended formulation is that already the initial ILP characterizes all feasible solutions, and we do

not have to iterate or to add cutting planes. Nevertheless, the formulations contain more variables, and the characterization with shape and consistency inequalities is less tight than the one obtained by the zero-one formulation.

5.4.1 Implementations

We provide implementations of the new algorithms for the label number maximization problem LNM. All algorithms, the visualization component, and the constraint graph data structure are implemented with LEDA, a library of efficient data types and algorithms (Mehlhorn and Näher, 1999). As described in Section 5.2, we determine the labeling model by changing the rules for constructing the set of boundary arcs. Our implementation of the constraint graphs provides this functionality so that we can easily choose among the six different labeling models in our algorithms. For the pure LP-based branch-and-bound algorithms we use the CPLEX library (CPLEX, 1999), our branch-and-cut algorithms are built with the ABACUS-framework (Jünger and Thienel, 2000). For the heuristic separation of positive cycles, we use the publically available code that is also used in the experimental comparison of shortest path algorithms (Cherkassky and Goldberg, 1999).

A preliminary computational evaluation shows that the best of the presented strategies is the iterative branch-and-bound scheme that rests upon the pure zero-one formulation. It avoids the hardness of the separation problem as well as the additional variables and constraints. For these reasons, we perform our experiments with Algorithm 5.1, and report on the results in the next section. We identify the following important parameters by experimenting with different settings.

Surprisingly, the choice how to model the fixed distance arcs is crucial for the running time of our algorithms. We proceed as illustrated in Figure 5.5 on page 124 and insert many small cycles, see also Remark 5.1 on page 123. Experiments with other possibilities, *e.g.*, one large cycle with all but one negative arc or all but one positive arc, result in increased running time due to a harder separation task. Furthermore, experiments with the branching order show that it is better to prefer the label separation arcs over the boundary arcs as candidates for branching variables.

Algorithm 5.2 *Iterative branch-and-bound scheme for the pure labeling problem*

Input: Initial ILP for the pure labeling problem LAB

Output: Solution for LAB or detect infeasibility

1: current ILP = initial ILP;
2: **repeat**
3: solve current ILP with branch-and-bound, stop when feasible solution found;
4: S = separated positive cycle inequalities;
5: **if** $S \neq \varnothing$ **then** add S to current ILP;
6: **until** $S \equiv \varnothing$ or current ILP infeasible

5.5 Computational Experiments

In this section we report on our computational results for the label number maximization problem. All results in this section are computed by the iterative branch-and-bound approach. We use a Sun Enterprise 450 with 1.1 Gigabyte main memory and two 400 MHz-CPUs for the experiments.

We test our integer linear programming-based approach on different types of instances in all six axis-parallel rectangular labeling models as defined in Figure 1.3 of the introduction.

Our first source of test data is publically available on the web page `http://www.math-inf.uni-greifswald.de/map-labeling/general/`. The site is maintained by Alexander Wolff and contains practical instances as well as randomly generated data. We test our implementation on several maps, *e.g.*, the German railway stations, see Figures 5.11-5.16, or maps of the United States of America, see Figure 5.17. The map of German railway stations (`german_railway_366.xyn`) contains 366 cities that have to be labeled with their names. We use a text font size of 8 pt. An optimal solution in the one-position model contains only 235 labels, see Figure 5.11. Already the two-position model (Figure 5.12) allows a placement of 59 more labels (294). The implementation places 311 labels in the one-slider model (Figure 5.13), 339 in the four-position model (Figure 5.14), and 349 in the two-slider model (Figure 5.15). Only 12 labels cannot be placed in an optimal solution for the four-slider model (354 labels, Figure 5.16).

The running times show a remarkable difference of the performance of our implementation in the different labeling models. Whereas it is very fast in the one-position, two-position, and one-slider models (below 13 seconds), the computation takes about seven and a half minutes in the two-slider model and half an hour in the four-slider model. In the four-position model, it takes the implementation more than two and a half days to find an optimal solution and prove its optimality. We defer a discussion of the reasons for the different running times to the end of this section.

For the example `us_abbrev_1041.xyn`, *i.e.*, the map of a major part of the United States of America, we scale the coordinates by a factor of 50 and use a text font size of 12 pt. Of the 1041 cities we can label 1004 in the four-position model (see Figure 5.17).

Additionally, we run our implementation on a set of practical data, the Munich drill hole instances from the above mentioned web page. The instances correspond to rectangular submaps of a map with 19,400 ground-water drill holes in the city of Munich with sizes n in the range $\{250, 500, 750, \ldots, 2750, 3000\}$. There are 30 instances of each size. The label sizes of these benchmark instances arise from heuristically scaling the labels to a large size for which still all the labels can be placed in the four-position model. Figure 5.18 shows such an instance with 500 drill holes. For these instances, we only compare the four-position and the four-slider model.

The plot in Figure 5.19 shows how long our implementation needs to place all the labels. It can be observed that the algorithm runs three to five times faster in the four-position model and is quite fast in both models. The good results indicate that our approach is suited to compute optimal solutions for large instances of the label size maximization problem. In addition, an implementation for LSM would benefit from the easier

Figure 5.11: Optimal solution of a map with German railway stations in the one-position model. 235 of 366 cities have labels with text font size 8pt, running time 2.76 s

algorithm for the pure labeling problem.

We also test our implementation on a widely used benchmark generator for randomly creating instances of labeling problems, according to the rules described in (Christensen *et al.*, 1995): First, we construct a set of n points with random coordinates in the range $\{0, \ldots, 792\}$ for the x- and $\{0, \ldots, 612\}$ for the y-coordinates. Each point-feature has a label of width 30 and height 7. Figure 5.20 shows a provable optimal solution with 795 labels in the four-slider model for such an instance with $n = 800$.

Following the scheme in (Verweij and Aardal, 1999) and (Verweij, 2000) we randomly generate 25 maps of size n with $n \in \{100, 150, \ldots, 950, 1000\}$. We limit the running time to 30 minutes of CPU-time for each component. Typically, only one or two difficult components exist, depending on the density of the instance. Table 5.2 shows for how many of the larger instances the computation of all components terminates within the time limit. Up to a size of 600 labels the implementation provides an optimal solution in all models in short computation time.

Figure 5.12: Optimal solution of a map with German railway stations in the two-position model. 294 of 366 cities have labels with text font size 8pt, running time 12.73 s

	Number of solved instances/size								
Model	600	650	700	750	800	850	900	950	1000
1P	25	25	25	25	25	25	25	25	25
2P	25	25	25	25	22	21	15	5	-
4P	24	18	3	-	-	-	-	-	-
1S	25	25	25	25	25	25	24	19	12
2S	25	24	22	19	11	1	-	-	-
4S	25	23	20	13	3	1	-	-	-

Table 5.2: Number of optimally solved instances for the randomly generated data. The first row describes the size of the instances, the following rows show how many instances our implementation solves to optimality within 30 minutes CPU-time in the respective model

The one-position model is the only model for which our implementation produces optimal solutions for all 475 instances within the time limit. Looking at discrete and slider

Figure 5.13: Optimal solution of a map with German railway stations in the one-slider model. 311 of 366 cities have labels with text font size 8pt, running time 4.12 s

models independently, we observe that the more restricted a model is, the easier is the computation. In general, the discrete models are easier to solve with the exception of the four-position model: Instances of the four-position model seem to be hardest for our implementation as already indicated by the railway station examples.

Figure 5.21 shows the percentage of labels that can be placed in each model for the "poor" models (one-position, two-position, and one-slider). We observe a linear decrease in the number of placed labels in relation to the total number of labels.

The same plot for the "good" models (four-position, two-slider, and four-slider) in Figure 5.22 shows a different behavior. In all models, the optimal solutions are quite close to the total number of labels, and their order reflects the freedom a label has in the respective model. However, the percentage of placed labels decreases quickly after an initial plateau which is close to 100%. Surprisingly, the trend of decreasing percentage seems to change for the larger instances. These anomalies in the four-slider and four-position model have the following reason: Only a few easy instances can be solved of that size, *i.e.*, instances which allow a relatively large number of placed labels.

Figure 5.14: Optimal solution of a map with German railway stations in the four-position model. 339 of 366 cities have labels with text font size 8pt, running time 65 h, 26 min, 23 s

We compare the four-slider model with all other models in Figure 5.23 which shows how many more labels can be placed in the four slider model as compared to the five other models. The plots indicate that the advantage of the slider models over the discrete models is considerable.

Figure 5.24 shows the running times for the randomly generated instances in the discrete models with a logarithmic scale. After a certain instance size, the running times increase rapidly. For the one-position model, the threshold of the "exponential explosion" is at about 2,500 labels and is not reached by the sizes of the test data. Again, the anomalies in the plots are due to the fact that easy instances are also faster to solve.

For the slider models, the data look similar, see Figure 5.25. The bends in the plots exist for the same reason as for the discrete models. It can be seen that for the randomly generated data the four-slider and four-position model have about the same running time behavior.

We conclude the chapter on map labeling by looking at the reasons why the performance of our approach depends such heavily on the labeling model. We can determine

Figure 5.15: Optimal solution of a map with German railway stations in the two-slider model. 349 of 366 cities have labels with text font size 8pt, running time 437.64 s

two main factors which influence the running time of our algorithm: On the one hand, this is the number of labels that cannot be placed in an optimal solution, *i.e.*, the difference $|\Lambda| - |\Lambda_P|$. In general, an optimal solution that allows a large number of placed labels is faster to compute since the number of possibilities for the set of removed labels is much smaller than in a solution in which many labels cannot be placed.

On the other hand, the tightness of the inequalities has an impact on the running time; the more restrictions on the variables, the faster the algorithms. Figures 5.24 and 5.25 reflect this order of tightness; the faster models are the more restricted ones.

Both factors, however, interrelate: In the more restricted models we can also place a smaller number of labels. This explains also the data in Table 5.2: in general, the inequalities are tighter for the discrete model, but in large instances the growing difference $|\Lambda| - |\Lambda_P|$ becomes more influential.

Figure 5.16: Optimal solution of a map with German railway stations in the four-slider model. 354 of 366 cities have labels with text font size 8pt, running time 1802.47 s

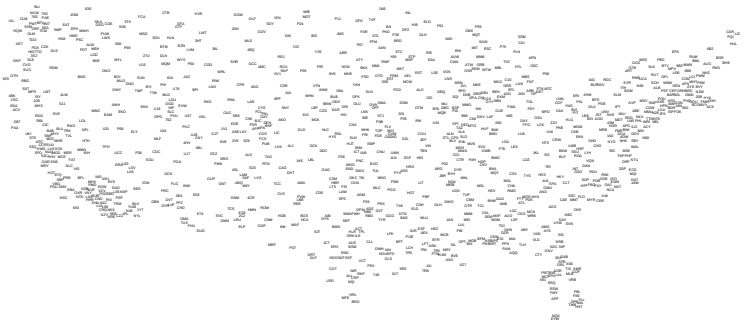


Figure 5.17: Optimal labeling for a map of the United States of America without Alaska and Hawaii, 1004 of 1041 cities have labels. Instance `us_abbrev_1041.xyn`, coordinates multiplied by 50, font size 12 pt, four-position model, running time 1317.31 s

Figure 5.18: 500 labeled ground-water drill holes in Munich

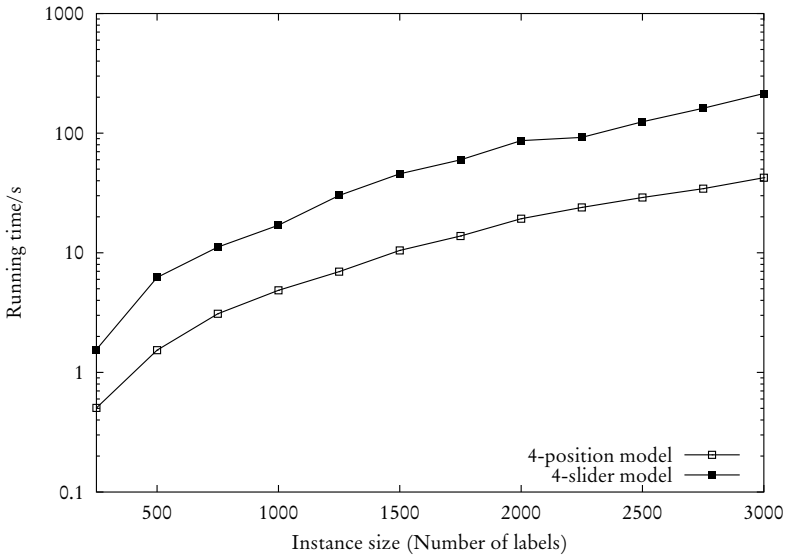


Figure 5.19: Running times for the Munich drillholes instances (logarithmic scale)

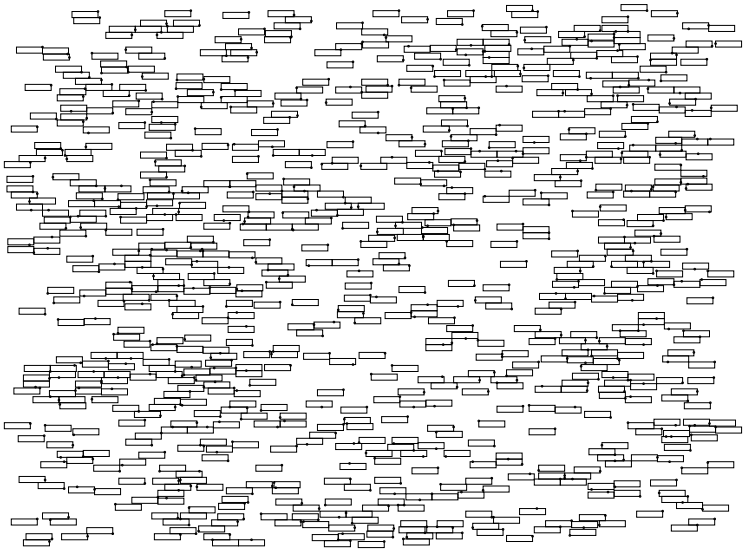


Figure 5.20: A provable optimal solution of an instance with 800 point-features generated according to the rules in (Christensen *et al.*, 1995). 795 points receive labels in the four-slider model, 40 minutes running time

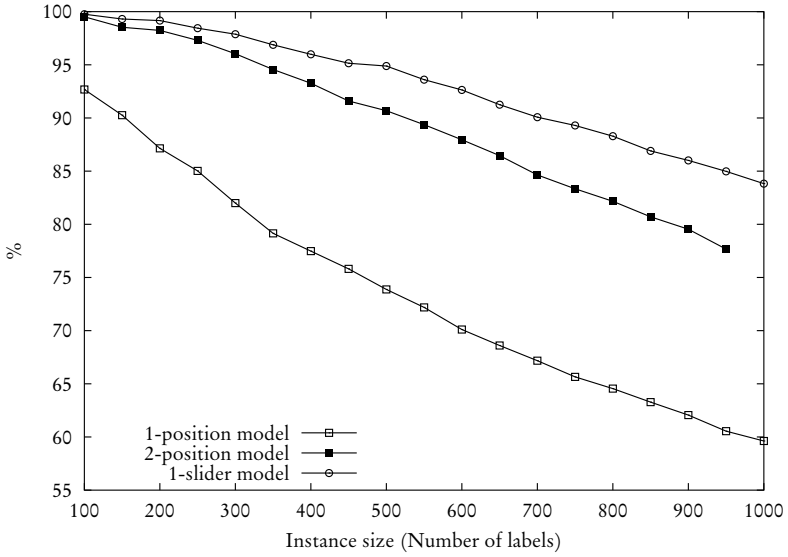


Figure 5.21: Percentage of labels that can be placed within each model: One-position model, two-position model, and one-slider model

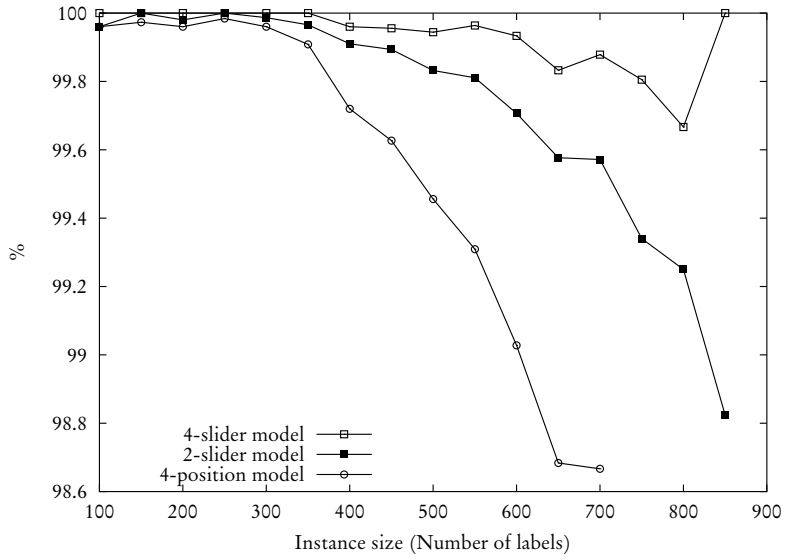
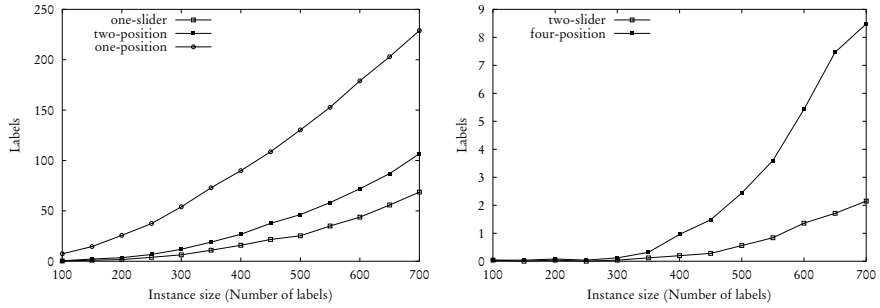


Figure 5.22: Percentage of labels that can be placed within each model: Four-Position, two-slider, and four-slider model



(a) Compared to 1P, 2P, and 1S (b) Compared to 2S and 4P

Figure 5.23: Number of more labels that can be placed in the four-slider model in comparison to other models

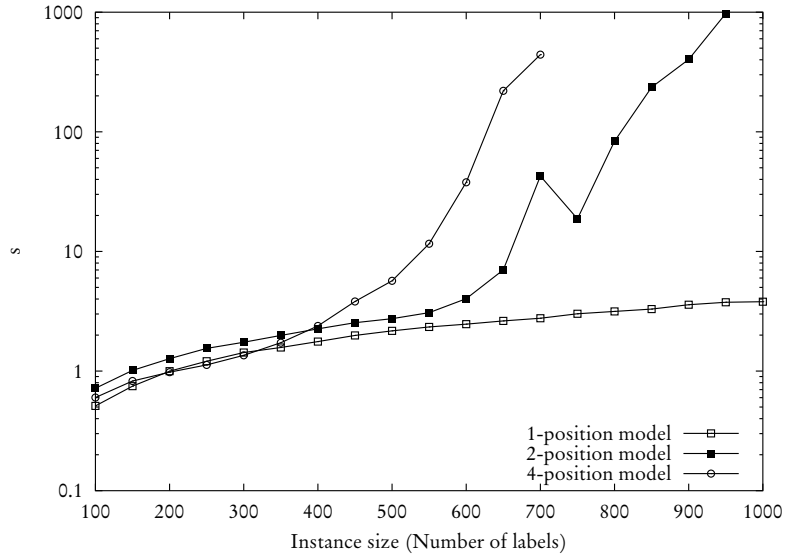


Figure 5.24: Running times for the randomly generated instances, discrete models, logarithmic scale

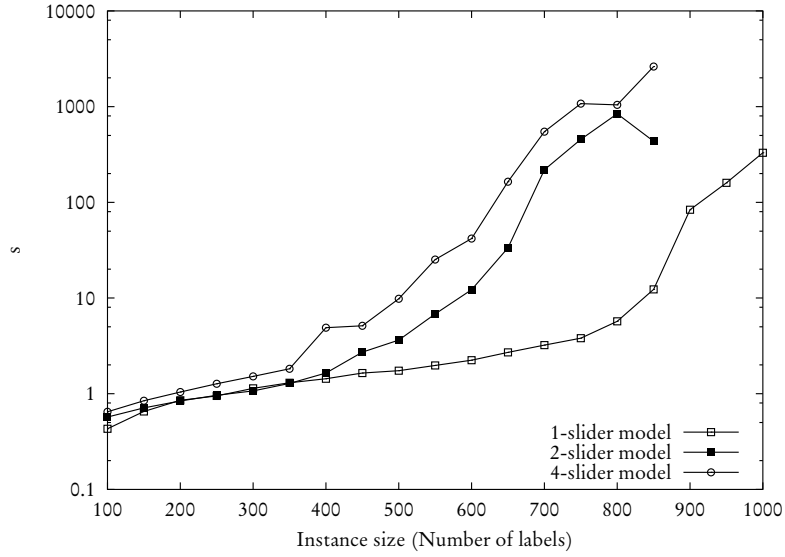


Figure 5.25: Running times for the randomly generated instances, slider models, logarithmic scale

Chapter 6

Application to a Graph Labeling Problem

This chapter combines our results from the two previous chapters in order to provide an algorithm for a combined drawing and labeling problem. We define *graph labeling problems* in Section 6.1 as graph drawing problems in which subsets of vertices and edges have to be labeled. Little research work exists on this subject.

In Section 6.2 we focus on a special graph labeling problem that arises in the area of automation engineering: drawing labeled state diagrams. We propose an approach that combines the last phase within the topology-shape-metrics scheme in orthogonal graph drawing with a labeling strategy and introduce the *combined compaction and labeling problem*. In Section 6.3 we merge the results of the previous chapters and obtain an equivalent combinatorial formulation of the problem. We present an integer linear programming formulation for the combinatorial version and conclude by presenting an exact algorithm for the combined compaction and labeling problem. To our knowledge this is the first exact algorithm especially designed to solve graph labeling problems.

6.1 Graph Labeling

Combinations of graph drawing and map labeling problems yield challenging mathematical problems and have direct applications, *e.g.*, the drawing of schematic maps such as subway maps or the automatic layout of state diagrams in automation engineering. We will provide a more detailed description of the latter problem in the next section. Unlike in map labeling where the position of the objects is specified in the input, the coordinates of vertices and edges in an instance of a graph drawing problem have yet to be determined and thus create additional degrees of freedom.

We call the class of graph drawing problems where subsets of vertices and edges have to be labeled *graph labeling problems*. Clearly, these combined problems are computationally at least hard as the included subproblems. Two straightforward approaches exist:

– *First draw, then label.* One way to attack a graph labeling problem is to divide it into the two subproblems drawing and labeling. Algorithms in this category first produce a graph drawing for the underlying graph, disregarding the labeling information. As a second step they solve a classical map labeling problem, *i.e.*, they treat vertices and edges of the drawing as point- and line-features which have to receive labels. Kakoulis and Tollis (1997a) present such an approach and consider,

in particular, edge labeling, see also (Kakoulis and Tollis, 1997b). Yet, the techniques assume an existing graph drawing as input.

The drawback of this approach is that, in general, the empty space of the drawing does but suffice to admit a feasible labeling for all labels. In this case and provided that all labels have to be placed, a strategy must choose among four unattractive possibilities:

1. It places the labels so that they overlap other labels or vertices and edges of the drawing. The last-placed labels may cover important parts of the drawing or crucial information in formerly placed labels.
2. It places some labels far away from the corresponding vertices or edges which makes it difficult for the reader of the drawing to associate the information in the labels with the appropriate objects.
3. It scales the drawing so that all labels can be placed. This decreases the quality of the graph drawing due to longer edges and larger drawing area.
4. It solves the label size maximization problem for the instance. The legibility of the labeling suffers from decreasing the scale factor too much.

Since the aesthetic criteria for graph labeling problems combine those for graph drawing and map labeling, the four possibilities can lead to a dramatic decrease in the solution quality.

– *Treat labels as vertices.* A second idea is to treat the whole problem as a graph drawing problem. Labels are modeled as vertices of prescribed size and attached to their corresponding vertices or edges by additional artificial edges and vertices. Then, a graph drawing algorithm which can handle vertices of fixed, prescribed size produces a drawing which results in a solution of the graph labeling problem by removing the artificial vertices and edges.

  Approaching a graph labeling problem in this manner may cause several problems. First, modeling the labels as vertices increases the vertex degrees of the underlying graph. The quality of many graph drawing algorithms depends heavily on the vertex degrees. Furthermore, only a few algorithms exist which can cope with vertices of prescribed size (see also Chapter 7 for this issue). A second disadvantage is that the additional artificial vertices and edges destroy structural properties of the underlying graph.

For certain applications, these approaches are successful, but in most cases the quality of the solutions is too bad to apply the above presented techniques in practice.

6.2 The Combined Compaction and Labeling Problem

In this section, we present a graph labeling problem which occurs in the area of automation engineering. After an informal description of the problem we show that the strategies

described in the last section do not produce satisfactory solutions. Our new approach combines the compaction problem of the topology-shape-metrics scheme for orthogonal graph drawing with the pure labeling problem. This enables us to apply the results from Chapter 4 (compaction) and Chapter 5 (labeling).

State diagrams are used for designing and running control systems like, *e.g.*, production controls or robot controls in the area of automation engineering. Figure 6.1 illustrates a typical hand-drawn state diagram of a control system.

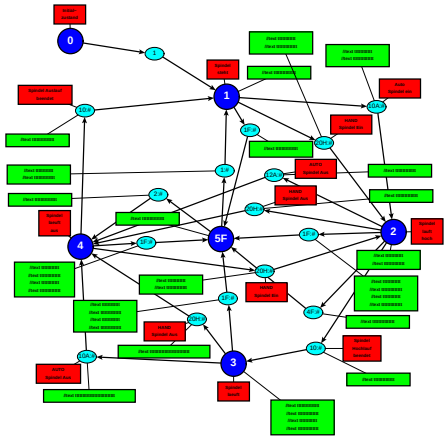


Figure 6.1: A hand-drawn state diagram, courtesy Siemens AG

A state diagram describes all possible states of a control system and the state transitions. A state can be, *e.g.*, the initial state, an error state or states like "motor is running" and is represented by a *state node* in the diagram. A transition from a state A to state B is displayed as an arc from state node A to state node B via a *transition node*. Rectangular labels which are attached to the state and transition nodes contain further information—in many cases several lines of program code.

The information in the drawing of Figure 6.1 is very hard to extract, as many lines cross each other, labels often overlap with transition lines and many labels are placed far away from their associated nodes.

The usage of scaling is restricted in this special problem since the size of the labels is fix and the labeled drawing must fit on a sheet of paper. The first approach in the previous section fails due to the large number of labels in state diagrams. The space in the drawing suffices only for a small fraction of the labels making it impossible to avoid overlaps. Also the second approach is not suitable for the layout of labeled state diagrams. Due to the bounded degree of the underlying graph in instances of this graph labeling problem, orthogonal graph drawing techniques yield layouts of particularly good quality. Modeling the labels as vertices leads to a loss of these underlying orthogonal structures and, as a consequence, to an increase in the drawing area.

As mentioned above, state diagrams are very well suited to apply orthogonal drawing

techniques. We propose to use the topology-shape-metrics scheme for the graph drawing part of the problems since it leads to the best orthogonal layouts in practice (Di Battista *et al.*, 1997). Yet we are aware that the appropriate space for the labels should already be reserved when determining the coordinates. This observation suggests a union of the compaction phase with a labeling technique. The overall approach is as follows: We determine a simple orthogonal representation for the underlying graph of a labeled state diagram with the planarization method described in Section 1.1.1.[1] The task is now to find coordinates for the vertices and bends of the state diagram, and to assign positions to the labels so that the output is an orthogonal grid drawing for the underlying graph combined with an overlap-free, unscaled labeling. We choose the most unrestricted labeling model, the four-slider model, in order to have the maximum freedom for the labeling subtask.

We formally define a feasible solution of the problems as follows:

Definition 6.1 (Labeled orthogonal grid drawing). Let H be an orthogonal representation of a four-planar graph $G = (V, E)$, let Λ be a set of labels of width and height $w, h : \Lambda \to \mathbb{Q}_+$, and let $a : \Lambda \to V$ denote the vertices that receive the labels. A labeled orthogonal drawing Γ_L satisfies the following properties.

(a) Γ_L is an orthogonal grid drawing of H.

(b) Each label is represented by a rectangle of width $w(\lambda)$ and height $h(\lambda)$ which contains $\Gamma_L(a(\lambda))$ on its boundary.

(c) A label $\lambda \in \Lambda$ does neither overlap nor include other labels or parts of the drawing.

Among the labeled orthogonal drawings we prefer those that have short edges due to the same reasons as for pure drawings. This leads to the following problem formulation:

Definition 6.2 (Combined Compaction and Labeling Problem, COLA). Let H be a simple orthogonal representation H of a four-planar graph $G = (V, E)$ with additional label information Λ, w, h, and a. We refer to the five-tuple (H, λ, w, h, a) as an instance of the combined compaction and labeling problem. Given such an instance, find a labeled orthogonal grid drawing Γ_L of H of minimum total edge length.

From a computational-theoretic point of view, the combined compaction and labeling problem is a difficult problem. Already the two subproblems, compaction and labeling are *NP*-hard (see Sections 4.1 and 5.1). subproblems.

6.3 An Exact Algorithm for the Combined Compaction and Labeling Problem

In this section we develop an exact algorithm for the COLA problem and proceed as in the previous chapters. An equivalent combinatorial formulation involving a pair of constraint graphs, the shape and labeling graphs, enables us to present an integer linear programming formulation. We propose to solve the ILP formulation with a branch-and-bound algorithm.

[1] Most of the underlying graphs of state diagrams are planar.

6.3.1 Unifying the Combinatorial Formulations

In order to combinatorially characterize the combined problem, we merge our combinatorial formulations for the two-dimensional compaction problem and the labeling problem. We will define the *shape and labeling graphs* which combine the elements of the respective pairs of constraint graphs for the subproblems. We adapt the notion of completeness and establish a one-to-one correspondence between solutions of a combinatorial problem version and labeled orthogonal drawings.

Definition 6.3 (Shape and labeling graphs). A pair of *shape and labeling graphs* consists of two constraint graphs $D_x = (N_x, A_x, w_x)$ and $D_y = (N_y, A_y, w_y)$ and corresponds to an instance $I = (H, \Lambda, w, h, a)$ of the combined compaction and labeling problem. Let $G = (V, E)$ be the underlying four-planar graph. The node sets of D_x and D_y are

$$\begin{aligned} N_x &= \mathcal{S}_v \cup \{l_\lambda \mid \lambda \in \Lambda\} \cup \{r_\lambda \mid \lambda \in \Lambda\} \\ N_y &= \mathcal{S}_x \cup \{b_\lambda \mid \lambda \in \Lambda\} \cup \{t_\lambda \mid \lambda \in \Lambda\} \ . \end{aligned}$$

The arc sets of the constraint graphs are

$$A_x = \{(\text{vert}(v), \text{vert}(w)) \mid (v,w) \in \vec{E}_r\} \cup \bigcup_{\lambda \in \Lambda} \{(l_\lambda, r_\lambda), (r_\lambda, l_\lambda), (l_\lambda, r_{a(\lambda)}), (l_{a(\lambda)}, r_\lambda)\}$$

$$A_y = \{(\text{hor}(v), \text{hor}(w)) \mid (v,w) \in \vec{E}_u\} \cup \bigcup_{\lambda \in \Lambda} \{(b_\lambda, t_\lambda), (t_\lambda, b_\lambda), (b_\lambda, t_{a(\lambda)}), (b_{a(\lambda)}, t_\lambda)\}.$$

Arcs between segments have unit weight, and arcs between segments and limits of labels have zero weight. The arcs between limits of labels are equal to the label size arcs and have also the same weights which depend on the width and height of the corresponding label.

Basically, the new pair of constraint graphs is a pair of shape graphs for the underlying orthogonal representation with additional nodes and arcs. Each label gives rise to four nodes which correspond to its limits. Furthermore, the arcs sets of the shape and labeling graphs include the label size arcs and a variant of the proximity arcs. The difference to the proximity arcs in map labeling problems is that the labels are linked to the segments of the corresponding vertices instead of fixed points in the plane. By definition, an instance I of the combined compaction and labeling problem gives rise to a unique pair of shape and labeling graphs which we will refer to by $\gamma(I)$.

Note that the notion of limits builds the link between the two different problems compaction and labeling. Substantially, a segment of the compaction problem and a label are modeled in the same way. Both are bounded by four limits for which nodes in the shape and labeling graphs exist. In principle, we can construct each solution of the combined compaction and labeling problem by assigning values to these nodes. Again, the rest of this section is dedicated to identifying properties of the pair of constraint graphs D_x and D_y so that feasible assignments for D_x and D_y lead to solutions of the combined compaction and labeling problem.

As for the shape graphs, we define an *extension* of the pair of constraint graphs $\gamma(I)$ as supergraphs which result from adding arcs. We know by the previous chapters that feasible assignments for these supergraphs are also feasible for each of their subgraphs.

As a first step, we extend the notion of completeness. Due to the additional label size arcs we must exclude positive cycles in the constraint graphs explicitly, since half of these arcs have a negative weight.

Definition 6.4 (Completeness of extensions of shape and labeling graphs). An extension of a pair of shape and labeling graphs is *complete* if and only if

(a) Both graphs do not contain positive cycles;

(b) For all pairs of nodes (i, j) one of the four following positive paths is contained in the arc sets:

$$r_i \xrightarrow{+} l_j \qquad r_j \xrightarrow{+} l_i$$
$$t_i \xrightarrow{+} b_j \qquad t_j \xrightarrow{+} b_i \ .$$

In the next section we show the one-to-one correspondence between complete extensions of shape and labeling graphs and labeled orthogonal grid drawings by relating feasible solutions of an integer linear program to feasible solutions of the combined compaction and labeling problem.

6.3.2 Integer Linear Programming Formulation

The similarity to the combinatorial formulations of the subproblems motivates the search for a complete extension of the shape and labeling graphs as in our approach to the two-dimensional compaction problem in Chapter 4. In order to separate unseparated labels and segments we consider potential additional arcs between the limits of these objects. Let $O = \mathcal{S} \cup \Lambda$ contain the objects which have to be separated. We can separate each unseparated pair of objects $\{o, p\} \in \binom{O}{2}$ by adding one of the arcs

$$A_{\text{sep}}(o, p) = \{(r_o, l_p), (r_p, l_o), (t_o, b_p), (t_p, b_o)\} \ .$$

Yet, for the combined problem it is difficult to bound the size of the potential arcs as successful as for the subproblems. Since the coordinates of the vertices are not fixed, we cannot determine the label separation arcs according to the overlapping regions as in the static case. However, two labels which must be placed in different faces of the drawing can never overlap and we do not have to ensure their separation. Even if this is not the case we can in many cases restrict the number of possible label separation arcs for a label pair λ and μ, *e.g.*, if the appropriate vertices $a(\lambda)$ and $a(\mu)$ share the same segment. To avoid the interaction between two segments I and J we proceed as in Chapter 4 and compute the sets $A_{\text{sep}}(I, J)$ by computing the maximal unique completion of the shape graphs $\sigma(H)$.

The following is a general integer linear programming formulation for the combined compaction and labeling problem. Let $A_x^{\mathcal{S}}$ and $A_y^{\mathcal{S}}$ be the arc sets in the pure shape graphs, and let

$$A_{\text{pot}} = \bigcup_{\{o,p\} \in \binom{O}{2}} A_{\text{sep}}(o, p)$$

be the set of potential arcs. We distinguish between the horizontal and vertical potential arcs and refer to them by A_{xpot} and A_{ypot}, respectively. We combine the formulation for total edge length minimization with the extended version for the pure labeling problem.

$$\min \quad \sum_{(I,J)\in A_x^S \cup A_y^S} c_J - c_I \tag{ILP.7}$$

$$\text{subject to} \quad \sum_{a\in A_{\text{sep}}(o,p)} x_a \geq 1 \qquad \forall A_{\text{sep}}(o,p) \neq \varnothing \tag{ILP.7.1}$$

$$c_j - c_i \geq w_x(a) \qquad \forall a = (i,j) \in A_x \tag{ILP.7.2}$$

$$c_j - c_i \geq w_y(a) \qquad \forall a = (i,j) \in A_y \tag{ILP.7.3}$$

$$c_j - c_i - (M_a + w_x(a))x_a \geq -M_a \qquad \forall a = (i,j) \in A_{\text{xpot}} \tag{ILP.7.4}$$

$$c_j - c_i - (M_a + w_y(a))x_a \geq -M_a \qquad \forall a = (i,j) \in A_{\text{ypot}} \tag{ILP.7.5}$$

$$0 \leq x_a \leq 1 \qquad \forall a \in A_{\text{pot}} \tag{ILP.7.6}$$

$$x_a \in \{0,1\} \qquad \forall a \in A_{\text{pot}} \tag{ILP.7.7}$$

We convert a solution of (ILP.7) into a labeled orthogonal grid drawing in a similar manner as for the subproblems and let the coordinate vector determine the assignments in the two constraint graphs. Let the constants M_a be defined as in the respective subproblems. Lemmas 4.16 and 5.5 state that the choices of potential arcs that correspond to feasible solutions of the ILP do not induce positive cycles in the shape and labeling graphs.

Theorem 6.1. *Let $I = (H, \Lambda, w, h, a)$ be an instance of the* COLA *problem. A feasible solution (x, c) of (ILP.7) for I corresponds to a solution Γ_L of the* COLA *problem and vice versa. The value of the objective function is equal to the total edge length in Γ_L.*

Proof. We perform the proof in two parts. First, we demonstrate that Γ_L is an orthogonal grid drawing for H. In a second step, we show that the resulting labeling is feasible for the drawing of H.

To prove the first part, it suffices to observe that (ILP.7) contains the integer linear programming formulation for the two-dimensional compaction problem. Hence, by Theorem 4.6, every feasible solution of the above ILP gives rise to an orthogonal grid drawing for H.

The above integer linear program also includes the extended ILP formulation for the pure labeling problem with the exception of the boundary arcs. However, due to the existence of separation arcs between label boundaries and segments, we can omit the boundary arcs. The placement of label λ that includes vertex $a(\lambda)$ is not possible, since it implies an intersection of a boundary of λ with an incident segment to $a(\lambda)$. The second step of the proof follows by Theorem 5.6. ∎

6.3.3 Branch-and-Bound Algorithm

We use the ILP formulation of the previous section in order to devise a branch-and-bound algorithm for the combined compaction and labeling problem. The algorithm is similar as

for the pure compaction task. Let $I = (H, \Lambda, w, h, a)$ be an instance of the graph labeling problem. We start by constructing the combined shape and labeling graphs $\gamma(I)$. By definition, the shape graphs $\sigma(H)$ are subgraphs of $\gamma(I)$. We consider the pair $\sigma(H)$ and compute its maximal unique completion with Algorithm 4.5. As for the pure compaction problems this results in the sets A_{sep} containing the potential arcs for those segment pairs that are not uniquely separable. In a following step, we determine the sets A_{sep} for pairs of labels and label-segment pairs.

With this information, we build the integer linear program (ILP.7) that can be solved with a similar LP-based branch-and-bound strategy as described in Section 4.3.3.

Unlike its subproblems, it seems that the combined compaction and labeling problem is not only difficult from a theoretical but also from a practical point of view. It turns out that instances of the combined compaction and labeling problem are quite difficult to solve with our technique. A prototype of our algorithm is able to solve medium-sized instances (about 20 vertices and 30 labels) in reasonable computation time (about half a minute). However, the results are superior to applying map labeling algorithms to graph drawings.

Moreover, to our knowledge, the approach is the first algorithm especially designed to solve a graph labeling problem. A first step to improve the running time of the branch-and-bound algorithm could be to reduce the number of potential arcs and thus the number of variables in the ILP formulation. Furthermore, we are confident that the newly introduced concept of shape and labeling graphs is suited for further research that could lead to heuristic algorithms for the combined compaction and labeling problem.

Chapter 7

Discussion and Extensions

In this thesis we develop a constraint-graph based approach to orthogonal placement problems that leads to first exact algorithms for many of the problem variants under consideration. Our new techniques are based on integer linear programming formulations for combinatorial formulations that are equivalent to the original problems. We suggest a decomposition of the orthogonal placement problems into a horizontal and vertical problem component and introduce a constraint graph for each component. We investigate combinatorial properties of these pairs of constraint graphs and prove that separate assignments of values to the nodes of these graphs lead to feasible solutions of the original problem if and only if the pair of graphs satisfies a certain path- and cycle-based property which we refer to as *completeness*. The separate computation of feasible values for constraint graphs is computationally easy and strongly related to minimum cost-flow and shortest path problems. Our combinatorial problem versions consist of finding the best set of certain arcs to add to a problem-specific initial pair of constraint graphs so that the resulting pair of graphs is complete and admits optimal separate assignments of values to the nodes. Given such an assignment it is straightforward to construct a solution of the original problem.

Compaction For the two-dimensional compaction problem in orthogonal graph drawing which appears in the third phase of the important topology-shape-metrics scheme, we are able to derive a new exact algorithm based on our constraint graph-based framework. The input of the compaction problem is an orthogonal representation that fixes the shape of the final drawing. We introduce a pair of constraint graphs, the *shape graphs*, which are in one-to-one correspondence to orthogonal representations. Based on the combinatorial properties of shape graphs we define a hierarchy of orthogonal representations: If already the shape graphs are complete, we can solve the compaction problem to provable optimality in polynomial time. We show that shape graphs of rectangular representations, which appear within constructive heuristics for the compaction problem, belong to this class. Furthermore, we provide a polynomial-time algorithm that checks for completeness and determines the maximal unique completion of the shape graphs. If the resulting pair of constraint graphs is complete—in this case we say that the corresponding orthogonal representation is *uniquely completable*—we also can solve the compaction problems to optimality in polynomial time. Otherwise, we identify the set of *potential arcs* which may lead to complete extensions. We provide several integer linear programming formulations for different variants of the compaction problems that model the choice of which

potential arcs to add to the maximally completed shape graphs and propose an LP-based branch-and-bound algorithm to determine optimal solutions of the corresponding two-dimensional compaction problem.

We also investigate a one-dimensional compaction scheme which is similar to compaction techniques known from VLSI design. We demonstrate that instances exist for which a linear number of alternating one-dimensional compaction steps is necessary. Moreover, we show that algorithms within the scheme do not approximate the compaction problem by a constant factor.

Our extensive computational study shows that the new exact algorithms can solve large instances of the compaction problems in short computation time. The longest running time of our implementation for instances of a widely used benchmark set of 11,582 graphs arising from real-world applications is ten seconds. We also provide the first experimental comparison of the state-of-the-art compaction heuristics in graph drawing and gain new and surprising insights by our computational results. Our experiments indicate that good compaction heuristics perform well on most instances of the compaction problem and that the high quality is not due to the constructive but to the improvement heuristics. Therefore, we propose the combination of a simple initial heuristics followed by a minimum-cost flow-based improvement step in a practical application. Moreover, we have learned that the implementation of algorithms can benefit considerably from experimental studies on a large number of instances from different sources. During the more than 500,000 compaction runs we still found a remarkable number of errors in our implementations which we considered stable in the beginning. In particular the pathological instances made us understand the two-dimensional compaction problems in depth and we are convinced that extensive experiments strongly contribute to the provision of good implementations.

Map Labeling We also apply the constraint graph-based technique to point-feature map labeling problems and obtain new exact algorithms in a similar manner as for the compaction problems. We consider six different axis-parallel rectangular labeling models and various optimization versions of the labeling problem with a special focus on the label number maximization problem. For the slider models, our algorithms are the first exact algorithms, and we find it remarkable that although the underlying problems are of continuous nature in this case, we can successfully attack large problem instances with a discrete approach.

Unlike the polytope that corresponds to our combinatorial versions of the compaction problems, we define the *labeling polytope* as a pure zero-one polytope. However, one of the inequality classes we use to describe integral points within that polytope, the *positive cycle inequalities*, can be exponential in size. Unfortunately, it turns out that the corresponding separation problem is *NP*-hard. Our proof shows that the equivalent *weight-constrained shortest cycle* problem is an *NP*-complete problem by reducing it to the *shortest weight constrained path* problem. Due to the hardness of the separation problem we present an extended integer linear programming formulation for labeling problems which evades the positive cycle inequalities at the cost of additional variables. The alternative formulation has the additional advantage of being very similar to the ILP formulation for the

compaction problem which facilitates the combination of the two approaches. Yet, our experiments show that the pure zero-one formulation is better suited to apply integer linear programming techniques. We evaluate an LP-based branch-and-bound algorithm for the extended formulation, a branch-and-cut algorithm, and an iterative branch-and-bound scheme for the label number maximization problem, and our experiments demonstrate that the iterative scheme is the best of the three approaches. Theoretically, this is due to the fact that the iterative scheme avoids the heuristic separation of the branch-and-cut algorithm and the additional variables of the pure branch-and-bound algorithm. Furthermore, the algorithm exploits the fact that we can solve the separation problem exactly in case of non-fractional LP-solutions. We provide extensive computational experiments in which we test our new algorithm for all six labeling models on a large set of benchmark data. The results show that the exact algorithms are competitive and solve large instances to provable optimality in reasonable computation time.

Graph Labeling Finally, we combine our approaches for two-dimensional compaction problems and labeling problems in order to provide a first exact algorithm for a problem in the class of graph labeling problems. As indicated above, the extended ILP formulation for the labeling problem is similar to the formulation for the compaction problem and we exploit the similarity of the approaches to create a common combinatorial framework for compaction and labeling problems.

Extensions

Already the fact that our exact algorithms for the three different problems compaction, map labeling, and graph labeling rest upon a common combinatorial framework demonstrates that our constraint graph-based techniques are very flexible. In the following we list several extensions of our approach that might prove useful in the realization of algorithms for problems as they occur in practice.

Compaction of Drawings in Other Orthogonal Models In the chapter on compaction we have assumed that the input is a simple orthogonal representation for a four-planar graph. Most of the graphs that occur in practice, however, do not have this property. Therefore, a variety of extensions have been proposed in order to extend the orthogonal drawing model for planar graphs with arbitrary vertex degree.

Fößmeier and Kaufmann (1996) introduce the KANDINSKY model in which vertices are mapped to a coarse grid and edges are mapped to paths in a finer grid. Several edges may leave a vertex at the same side resulting in parallel edges and 0° degrees.

In the case that there is only one common grid, drawing a vertex v as an orthogonal box cannot be avoided if the degree of v exceeds four. Many algorithms in the topology-shape-metrics scheme produce orthogonal drawings according to this standard with varying restrictions to the sizes of the boxes (Tamassia *et al.*, 1988; Fößmeier and Kaufmann, 1996, 1997; Klau and Mutzel, 1998; Bertolazzi *et al.*, 2000).

All orthogonal drawing models for high-degree graphs constrain the dissection process for constructive heuristics remarkably, leading to many different cases that have to be

considered in an dissection algorithm for these models. Our exact techniques have the advantage of not transforming the underlying shape into an auxiliary structure and they can be applied to any of the above models.

Among the most important models in practice are those which are able to cope with vertices of prescribed size. In many applications such as drawing entity-relationship diagrams or UML diagrams the sizes of the individual rectangles that represent the vertices is fixed in the input. Eiglsperger, Fößmeier, and Kaufmann (1999) and Di Battista, Didimo, Patrignani, and Pizzonia (1999a) use variations of the KANDINSKY model to handle this restriction, however, the number of bends in the drawings is comparatively high. Klein (2000) provides a heuristic integer linear programming-based method and compares different approaches to this problem.

We can use the constraint graph-based approach in the following manner to produce drawings with prescribed vertex sizes: First, we create an auxiliary drawing in which the sizes of the rectangles that represent vertices might differ from the values given in the input. However, it is easy to ensure that the width and height of each rectangle are large enough so that the "real" rectangle can be placed inside. The next step consists of routing the edges from border of the real rectangle to the border of the auxiliary rectangle. This process might create a large number of bends and we propose to use the underlying constraint graphs to remove many of these bends. Since this technique is applicable to general orthogonal drawings, we defer its explanation to the next paragraph.

An interesting further application of our constraint graph-based compaction techniques is to non-orthogonal grid drawings. We treat general grid drawings like orthogonal drawings and consider the two directions of the compaction separately: a sloped edge gives rise to both a horizontal and a vertical constraint. The application of our approach is particularly promising in case the number of sloped edges is low like, *e.g.*, in the quasi-orthogonal drawing model introduced in (Klau and Mutzel, 1998). In this drawing standard, sloped edges occur only locally around vertices of high degree.

Furthermore, it is possible to give each edge an individual weight in the objective function. In this manner, edges with higher values are considered more important and will preferably be assigned a shorter length. In VLSI-design, the weight factor is usually chosen according to the electric resistivity of the corresponding wire. Wires with high resistivity should be short in the resulting layout.

Removing Bends by Compaction A variety of heuristic post-processing techniques exist that aim at improving the quality of an orthogonal grid drawing not only with respect to area or edge length but also to the number of bends in the drawing (Tamassia and Tollis, 1989; Fößmeier *et al.*, 1998; Six *et al.*, 2000). We present a new technique to reduce the number of bends in an orthogonal grid drawing which uses our constraint graph-based framework.

A crucial observation is that we can delete every pair of consecutive bends on the same edge if the resulting shape still admits an orthogonal grid embedding, see Figure 7.1. Our constraint-graph base approach is well suited to detect these situations. Let $\vec{E}_b$ be the set of half-edges with both endpoints corresponding to artificial vertices that represent bends. When constructing the shape graphs, we can set the weights of the arcs that correspond to

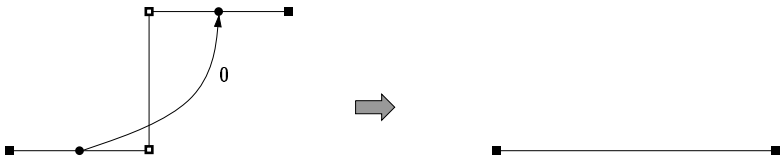


Figure 7.1: Removing bends with the constraint-graph based approach. Empty squares represent bends

half-edges in $\vec{E}_b$ to zero.

We can reduce the number of bends at the same time as minimizing the total edge length. In case minimizing the total number of bends is more important than short edges, we can assign higher objective function coefficients to the zero weighted arcs in the shape graphs.

If the reduction of total edge length and area is more important, we propose to temporarily insert virtual bends, thus adding more flexibility to the compaction process. With the above described technique we are able to remove many of the virtual bends. The remaining additional bends help to decrease the total edge length and area of the drawing.

The post-processing techniques are particularly powerful when applied in combination with our exact algorithms since many situations involve the interaction of the two dimensions. Yet, the techniques are not restricted to exact algorithms and can be applied in the same manner within the one-dimensional constraint graph-based improvement heuristics.

More General Labeling Models Practical map labeling applications use even more general labeling models as, *e.g.*, the four-slider or the four-position model. Furthermore, other models such as the three-position model are interesting from a theoretical point of view.

We can simulate general k-position models by introducing parallel proximity arcs with different weights according to k and further restricting the boundary arcs in case the k positions are not symmetric. *E.g.*, for the three-position model, we take the four-position model as a basis and prevent one of the possible placements by adding an appropriate inequality to the integer linear programming formulation.

Moreover, we can easily model different labeling models for different labels. The restrictions on the boundary arcs are local and the constant d that switches between discrete and slider models would be replaced by a vector $d \in \{1, 2\}^{\Lambda}$.

Preferred Label Positions and Labels of Different Importance In many practical applications of the label number maximization problem, some labels are more important than others. It is easy to integrate this into our approach: The objective function of the integer linear program changes to

$$\sum_{\lambda \in \Lambda} z_\lambda y_\lambda$$

and accordingly in the ILP in which the variables for labels have been substituted. The value z_λ denotes the importance of label λ. The algorithm will then prefer more important labels and remove less important ones more easily.

Another practically motivated extension is to model preferable positions of labels: Often, a label should be placed rather at its rightmost and upmost position than at other possible positions. We suggest to define a weight vector for the boundary arcs and incorporate it into the objective function.

Non-Touching labels The labeling model we use in Chapter 5 allows labels to touch. In most practical applications this is not desirable since touching labels decrease the legibility of a map. We suggest two strategies to ensure a minimum distance between two labels:

- Either all or only touching labels can be scaled down by a factor $1 - \varepsilon$.
- In our constraint graph-based approach, label separation arcs ensure that pairs of labels do not overlap. Nevertheless, they may touch, since the weights of these arcs is zero. This value can be changed to the desired minimum distance between labels which is the approach we have taken to produce Figure 1.1(c) on page 1, the example in the introductory chapter.

Further Work and Open Problems

It is certainly promising to deeper investigate the problem-specific polytopes in order to obtain a tighter characterization of the appropriate problems. In particular, the identification of facet-defining inequalities leads in many cases to considerably better bounds.

Our approach to compaction problems is located within the last phase of the topology-shape-metrics scheme and thus operates in a fixed embedding and fixed shape setting.[1] An interesting question is how the optimal drawings with respect to the aesthetic criteria change if the embedding or the shape or both may be changed.

Another generalization of orthogonal placement problems is to see them as constrained two-dimensional packing problems. A remarkable amount of research work exists for multi-dimensional packing problems including graph-theoretic approaches that exploit the relation to interval and comparability graphs. We would like both to learn from the formulations for packing problems and to investigate the possible application of our approach to multi-dimensional packing problems.

To date, no approximation algorithms exist for the compaction problems, and we show in Chapter 4 that heuristics based on one-dimensional compaction do not approximate the problems within a constant factor. It would be very interesting to have efficient heuristics with a good performance guarantee. Moreover, we believe that good heuristics for orthogonal placement and related problems can be build upon our constraint graph-based framework. These constraint graph-based heuristics might be candidates for approximation algorithms. In addition they could be applied at each node in the branch-and-bound or branch-and-cut trees in order to improve the global upper bounds.

[1] Note however, that the shape may be changed to a certain extent if we apply our compaction techniques to remove the number of bends.

Future work could explore the possibilities of our approach to related problem families. Promising areas are line and area feature labeling, edge labeling, labeling with obstacles, scheduling, two- and three-dimensional packing, and facility location. We are confident that our new combinatorial framework for orthogonal placement problems is flexible and powerful enough to build the basis of further successful and competitive combinatorial algorithms for orthogonal placement problems and related problem families.

Appendix

A Deutsche Zusammenfassung

Diese Arbeit analysiert zwei Problemfamilien aus dem Bereich der Informationsvisualisierung: Kompaktierung orthogonaler Gitterzeichnungen und Beschriftung von Punktmengen. Die betrachteten Kompaktierungsprobleme entstehen in der letzten Phase des sogenannten *Topology-Shape-Metrics*-Ansatzes, einer der wichtigsten Methoden im Bereich des orthogonalen Graphenzeichnens. Die Aufgabe besteht darin, eine gegebene dimensionslose Beschreibung der *orthogonalen Form* eines Graphen in eine orthogonale Gitterzeichnung mit kurzen Kanten und geringem Flächenverbrauch zu transformieren. Die zweite Problemfamilie, die in dieser Arbeit untersucht wird, ist dem Bereich Computer-Kartografie zuzuordnen. Hier ist die Aufgabe, eine gegebene Menge von rechteckigen *Labels* so zu platzieren, dass jedes Label einen zugehörigen Punkt berührt und gewisse Kriterien erfüllt werden, z. B. dass sich keine Labels überlappen. In einer klassischen Anwendung repräsentieren die Punkte beispielsweise Städte einer Landkarte, und die Labels enthalten die Namen der Städte.

Die oben genannten Platzierungsprobleme haben viele Gemeinsamkeiten: Sowohl Zeichnungen von Graphen als auch Landkarten übermitteln komplexe Information über Relationen zwischen Objekten in Form einer geometrischen Repräsentation. Darüberhinaus hängt der Nutzen dieser Repräsentation stark von der Qualität des Layoutprozesses ab. Das übergeordnete Ziel ist es in beiden Fällen, eine Zeichnung oder Beschriftung maximaler Lesbarkeit zu erstellen; das Resultat muss die unterliegende Information auf intuitive Art und Weise effektiv übermitteln. Abbildung A.1 zeigt "gute" und "schlechte" Lösungen für je ein Beispiel von orthogonalen Platzierungsproblemen.

Warum sind die beiden linken Platzierungen in Abbildung A.1 besser als die beiden rechten? Viele Kanten in der rechten Zeichnung des Graphen sind sehr lang, ohne dass ein Grund dafür erkennbar ist. Für den Benutzer der Zeichnung ist dies verwirrend. Zudem konnte die linke Zeichnung um 25% vergrößert werden und benötigt immer noch weniger Zeichenfläche als die rechte. Aufgrund der daraus resultierenden besseren Auflösung und der kürzeren Kanten ist sie der rechten überlegen. Beim Beschriftungsproblem sind die Kriterien noch offensichtlicher: In der rechten Karte fehlt Information, weil einige Label nicht platziert wurden. Zudem überlappen sich viele Label, und man kann den Text nicht lesen.

Eine weitere Gemeinsamkeit der hier betrachteten Probleme ist, dass sie, komplexitätstheoretisch gesehen, schwere Probleme sind. Eine ansprechende Zeichnung eines Graphen mit 20 Knoten von Hand zu erstellen ist schwierig und zeitaufwändig. Das gleiche gilt für das Beschriften von Landkarten mit vielen Labels. Sämtliche Probleme, die wir in dieser

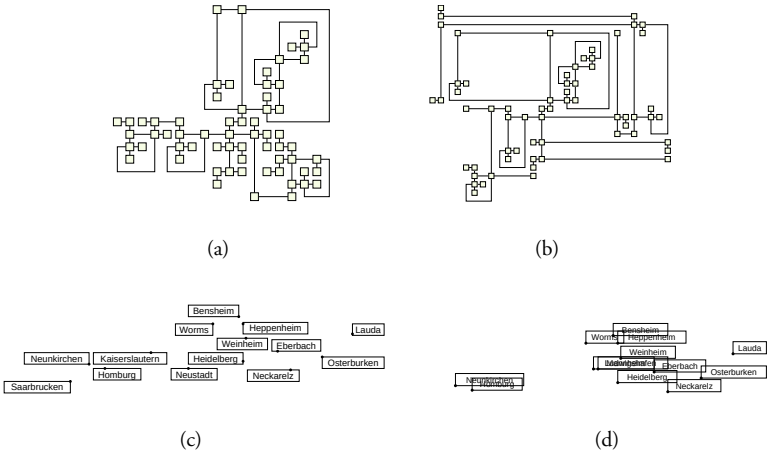


Abbildung A.1: Gute und schlechte Lösungen orthogonaler Platzierungsprobleme

Arbeit betrachten, gehören zur Klasse der *NP*–schwierigen Problemen.

In dieser Arbeit präsentieren wir ein gemeinsames theoretisches Gerüst für zweidimensionale Kompaktierungsprobleme und Beschriftungsprobleme. Die kombinatorische Formulierung ermöglicht es uns, exakte Algorithmen für die Probleme zu entwickeln, die große Probleminstanzen in kurzer Rechenzeit lösen können. Wir nutzen die Tatsache aus, dass orthogonale Platzierungsprobleme auf natürliche Art und Weise in zwei weitestgehend separate Problemkomponenten zerfallen, eine horizontale und eine vertikale Komponente. Zudem können wir jede Lösung durch Zuweisung von x– und y–Koordinaten an problemspezifische Objekte generieren; im Fall der Kompaktierungsprobleme sind dies die Knoten und Knicke des Graphen, im Fall der Beschriftungsprobleme sind es die Begrenzungen der rechteckigen Labels.

Unser neuer Ansatz für orthogonale Platzierungsprobleme basiert auf einem Paar von horizontalen und vertikalen *Constraint*–Graphen. Constraint–Graphen sind gerichtete Graphen, die Reihenfolgebeziehungen zwischen Objekten kodieren. Jedes Objekt ist im Graph durch einen Knoten repräsentiert. Zusätzliche Kantengewichte können diese Beziehungen noch spezifizieren. In orthogonalen Platzierungsproblemen korrespondieren die Objekte zu x– und y–Koordinaten der Knoten, Knicke oder Labelbegrenzungen; die gewichteten Kanten drücken Lagebeziehungen zwischen den Objekten aus. Die Idee des Constraint–Graph–Ansatzes besteht darin, die Koordinaten für die Originalprobleme durch Zuweisung von Werten an die Knoten der Constraint–Graphen zu bestimmen.

Constraint–Graphen wurden bereits in den Bereichen VLSI–Design und Scheduling benutzt. Insbesondere die Arbeit von Bartusch *et al.* (1988) weist einige Ähnlichkeiten mit unserem Ansatz auf. In Scheduling–Problemen repräsentieren die Knoten eines Constraint–Graphen Aufgaben und die gewichteten Kanten zeitliche Reihenfolge–Constraints zwischen den Aufgaben. Die Autoren betrachten die Optimierung über einer Menge von

zulässigen Schedules mit Zeit– und Ressourcen–Constraints. Sie charakterisieren diese Menge als Erweiterung einer partiellen Ordnung, die gewisse ordnungstheoretische Eigenschaften erfüllt. Aufgrund der eindimensionalen Struktur des Scheduling–Problems müssen keine Beziehungen zwischen verschiedenen Constraint–Graphen berücksichtigt werden. Diese stehen in unserem Ansatz im Mittelpunkt, da sie die Interaktion der Dimensionen in unseren zweidimensionalen Problemen modellieren.

Sowohl für die Kompaktierungs– als auch für die Beschriftungsprobleme führen wir spezielle, problemabhängige Paare von Constraint–Graphen ein: Im ersten Fall sind das die *Shape–Graphen*, im zweiten Fall die *Labeling–Graphen*. Wir studieren Eigenschaften dieser Paare und identifizieren eine pfad– und kreisbasierte kombinatorische Eigenschaft, die wir *Vollständigkeit* nennen. Diese Eigenschaft erlaubt es uns, die Umstände zu charakterisieren, unter denen jede zulässige separate Zuweisung von Werten an die Knoten jedes einzelnen Constraint–Graphs einer zulässigen Lösung des Originalproblems entspricht. Mit Hilfe des Konzepts der Vollständigkeit können wir äquivalente kombinatorische Umformulierungen der Originalprobleme aufstellen, das heißt, wir betrachten die ursprünglich geometrischen Probleme als graphentheoretische Probleme in einem Paar von Constraint–Graphen. Im Wesentlichen besteht die neue Aufgabe darin, eine Menge von zusätzlichen Kanten auszuwählen, die zu den Shape– oder Labeling–Graphen hinzugefügt werden kann, so dass das resultierende Paar von Constraint–Graphen vollständig ist. Falls die hinzuzufügende Menge für eine Instanz leer oder eindeutig ist (was wir in polynomieller Zeit ermitteln können), können wir die zugehörigen Originalprobleme in polynomieller Zeit beweisbar optimal lösen. In den meisten Fällen gibt es jedoch mehrere Möglichkeiten, und die Komplexität der kombinatorischen Probleme liegt darin, die richtige Teilmenge von zusätzlichen Kanten auszuwählen.

Nichtsdestotrotz sind die kombinatorischen Formulierungen sehr gut geeignet, um Methoden der ganzzahligen linearen Programmierung anzuwenden. Wir definieren problemspezifische Polytope, die der Menge der vollständigen Erweiterungen der Shape– oder Labeling–Graphen entsprechen und charakterisieren ganzzahlige Punkte in diesen Polytopen durch ganzzahlige lineare Programme. Wir entwickeln Branch–and–Bound– und Branch–and–Cut–Algorithmen und präsentieren umfassende experimentelle Studien. Unsere Resultate zeigen, dass wir große Instanzen der Platzierungsprobleme in kurzer Rechenzeit beweisbar optimal lösen können.

Ein Vorteil unseres gemeinsamen theoretischen Gerüstes für Kompaktierungs– und Beschriftungsprobleme ist, das wir sie relativ einfach kombinieren können, um Graphbeschriftungsprobleme zu lösen. Diese Klasse von Problemen vereint Charakteristiken der Gebiete *Graph Drawing* und *Map Labeling*, und nur wenige Forschungsarbeiten existieren in dieser Schnittmenge. Wir präsentieren ein neues Graphbeschriftungsproblem, das im Bereich der Automatisierungstechnik entsteht: das automatisierte Zeichnen und Beschriften von Zustandsdiagrammen. Wir kombinieren unsere Resultate für die beiden Teilprobleme, Kompaktierung und Beschriftung, und entwickeln einen exakten Algorithmus für das neue Problem. Unser Verfahren ist der erste exakte Algorithmus im Gebiet Graphbeschriftung.

Übersicht über die Arbeit

Kapitel 1 enthält eine Einführung in die Gebiete *Graph Drawing* und *Map Labeling* unter besonderer Berücksichtigung von orthogonalen Zeichenverfahren im Topology–Shape–Metrics–Ansatz und Beschriftungsverfahren für Punktmengen.

Das zweite Kapitel fasst die benötigten mathematischen Grundlagen zusammen: Wir beginnen mit Eigenschaften und Strukturen aus der Graphentheorie und gehen dann über die Theorie der planaren Graphen zum Bereich *Graph Drawing*. Viele der in dieser Arbeit entwickelten Verfahren sind kombinatorischer Natur und basieren auf Methoden der linearen Programmierung. Nach einer kurzen Einführung in kombinatorische Optimierung präsentieren wir generische Verfahren, die Optimallösungen für ganzzahlige lineare Programme berechnen.

Kapitel 3 ist den Constraint–Graphen gewidmet. Nach eine formalen Definition untersuchen wir, unter welchen Umständen eine zulässige Zuweisung von Werten an die Knoten eines Constraint–Graphen möglich ist. Die Verfahren in diesem Kapitel hängen stark mit minimalen Kostenflussproblemen zusammen und stellen in einer gewissen Weise den eindimensionalen und statischen Fall der Platzierungsprobleme dar. Wir präsentieren Verfahren, die für diese Teilprobleme optimale Zuweisungen in Bezug auf verschiedene Kriterien berechnen. Diese Methoden stellen die Basis unseres Ansatzes für die zweidimensionalen Probleme dar.

In Kapitel 4 betrachten wir Kompaktierungsprobleme im orthogonalen Graphenzeichnen. Nach einer formalen Einführung und der Präsentation von komplexitätstheoretischen Resultaten in Abschnitt 4.1, entwickeln wir das neue kombinatorisches Gerüst in Abschnitt 4.2. Wir führen die sogenannten *Placement–Graphen* als ein generisches Paar von Constraint–Graphen ein und definieren die Shape–Graphen, die in eineindeutiger Beziehung zu den Instanzen der Kompaktierungsprobleme stehen. Nach einer Definition des Konzepts der Vollständigkeit präsentieren wir ein zentrales Theorem, das vollständige Placement–Graphen mit zulässigen, und insbesondere optimalen, Lösungen der orthogonalen Kompaktierungsprobleme in Beziehung setzt. Wir besprechen detailliert existierende Konstruktions– und Verbesserungsheuristiken im Bereich Kompaktierung und deren Beziehungen zur neuen kombinatorischen Charakterisierung. Wir untersuchen insbesondere ein eindimensionales Kompaktierungsschema, das auch im Bereich VLSI–Design Anwendung findet und zeigen, dass Instanzen existieren, für die Algorithmen in diesem Schema eine lineare Anzahl von Iterationen benötigen. Ferner beweisen wir, dass Algorithmen in diesem Schema keine Kandidaten für Approximationsalgorithmen sind. Am Ende des Abschnitts präsentieren wir unsere kombinatorische Reformulierung des Kompaktierungsproblems und beweisen dessen Äquivalenz zum ursprünglichen Problem. Basierend auf der neuen Formulierung entwickeln wir in Abschnitt 4.3 exakte Algorithmen. Wir führen das Konzept von eindeutig vervollständigbaren Shape–Graphen ein und präsentieren einen exakten Polynomialzeitalgorithmus für diese Klasse. Der Algorithmus dient als Preprocessing–Schritt für den allgemeinen Fall, da er die Menge von zusätzlichen Kanten ermittelt, die Teil einer vollständigen Erweiterung sein können. Wir benutzen ein ganzzahliges lineares Programm, um aus dieser Menge von Kanten die besten auszuwählen, das heißt jene, deren Hinzufügen die Shape–Graphen so vervollständigt, dass eine optimale

Zuweisung von Werten an die Knoten in den Constraint–Graphen möglich ist. Wir geben einen Branch–and–Bound–Algorithmus an, der optimale Lösungen für das zweidimensionale Kompaktierungsproblem berechnet. Zudem identifizieren wir die Klasse von Kreisungleichungen, die eine bessere Beschreibung des problemspezifischen Polytops ermöglicht und zeigen, wie man das zugehörige Separierungsproblem in polynomieller Zeit lösen kann. Mit Hilfe der Separierung erweitern wir den Branch–and–Bound–Algorithmus zu einem Branch–and–Cut–Algorithmus, indem wir Schnittebenen an jedem Knoten im Branch–and–Bound–Baum berechnen. Das Ende des Kapitels enthält eine ausführliche experimentelle Studie, in der wir die state–of–the–art Kompaktierungstechniken vergleichen. Die Studie zeigt, dass wir das zweidimensionale Kompaktierungsproblem für Instanzen, wie sie in praktischen Anwendungen vorkommen, in kurzer Rechenzeit lösen können: Die längste Laufzeit unserer Implementierung für eine Instanz aus einer Menge von 11,582 aus der Praxis stammenden Benchmark–Graphen ist zehn Sekunden. Darüberhinaus zeigt unser Vergleich der Heuristiken, dass diese im Allgemeinen gute Lösungen für das Kompaktierungsproblem produzieren. Überraschenderweise ist die Qualität der Heuristiken nicht das Verdienst der Konstruktionsheuristiken, sondern der Verbesserungsheuristiken.

Das Thema des fünften Kapitels sind Beschriftungsprobleme. Wir entwickeln einen ähnlichen Ansatz wie für die Kompaktierungsprobleme und stellen kombinatorische Charakterisierungen verschiedener Beschriftungsprobleme unter sechs verschiedenen achsenparallelen Beschriftungsmodellen vor. Wir konzentrieren uns dabei hauptsächlich auf das Problem, die Anzahl der Labels in einer Beschriftung zu maximieren. Nichtsdestotrotz können wir unsere kombinatorische Formulierung auch für verwandte Probleme benutzen, z. B. die Ermittlung des maximalen Skalierungsfaktors, so dass eine überlappungsfreie Platzierung aller Labels möglich ist. Unsere Formulierung führt zu neuen Algorithmen, die große Instanzen der Probleme in kurzer Rechenzeit optimal lösen können. Abschnitt 5.1 enthält präzise Definitionen der Beschriftungsprobleme, präsentiert den Stand der Forschung auf diesem Gebiet und gibt einen umfassenden Überblick über die komplexitätstheoretischen Eigenschaften der Optimierungsprobleme, die beim Beschriften von Punktmengen entstehen. Wir definieren die Labeling–Graphen in Abschnitt 5.2. Diese Paare von Constraint–Graphen stehen in eineindeutiger Beziehung zu Instanzen der Beschriftungsprobleme, und wir benutzen sie, um äquivalente kombinatorische Formulierungen für die verschiedenen Optimierungsprobleme anzugeben. Basierend auf den neuen Formulierungen definieren wir das problemspezifische Polytop in Abschnitt 5.3 und präsentieren ganzzahlige lineare Programme, um die ganzzahligen Punkte des Polytops zu beschreiben. Wir untersuchen die Klasse der positiven Kreisungleichungen und zeigen, dass das zugehörige Separierungsproblem *NP*–schwierig ist. Um das Problem zu umgehen, geben wir auf Kosten einer erhöhten Anzahl von Variablen eine alternative Formulierung an. Abschnitt 5.4 enthält exakte Algorithmen für verschiedene Beschriftungsprobleme: Wir entwickeln Branch–and–Bound– und Branch–and–Cut–Algorithmen, sowie ein iteratives Branch–and–Bound–Schema. Für die kontinuierlichen Beschriftungsmodelle sind dies die ersten exakten Algorithmen, und wir finden es bemerkenswert, dass unser diskreter Ansatz optimale Lösungen für ein kontinuierliches Problem produziert. Es stellt sich heraus, dass der iterative Ansatz am besten geeignet ist, große Instanzen zu lösen, da er zum einen die zusätzlichen Variablen und zum anderen das *NP*–schwierige Separie-

rungsproblem vermeidet. Wir nutzen aus, dass die Separierung von verletzten positiven Kreisungleichungen in polynomieller Zeit möglich ist, wenn die zugehörige LP–Lösung nicht fraktional ist. Das Ende des Kapitels enthält umfassende experimentelle Resultate für eine große Zahl von Benchmark–Instanzen. Die Resultate zeigen, dass unser neuer Ansatz auch für große Instanzen in zumutbarer Rechenzeit beweisbar optimale Lösungen produziert.

In Kapitel 6 wenden wir die Resultate von den beiden vorangehenden Kapiteln auf ein neues Problem aus der Klasse der Graphbeschriftungsprobleme an. Wir diskutieren Eigenschaften dieser Problemklasse in Abschnitt 6.1 und konzentrieren uns in Abschnitt 6.2 auf ein spezielles Graphbeschriftungsproblem, das automatische Zeichnen und Beschriften von Zustandsdiagrammen. In Abschnitt 6.3 entwickeln wir einen exakten Algorithmus für dieses Problem, der auf einer Kombination der theoretischen Gerüste für die Teilprobleme Kompaktierung und Beschriftung basiert. Unser Algorithmus ist der erste exakte Algorithmus für Graphbeschriftungsprobleme.

Das siebte und letzte Kapitel enthält abschließende Bemerkungen und gibt einige mögliche Erweiterungen des neuen kombinatorischen Ansatzes an. Wir zeigen, dass der Ansatz geeignet ist, eine Reihe von praktischen Anforderungen an Kompaktierungs– und Beschriftungsprobleme elegant miteinzubeziehen und führen verwandte Gebiete wie z. B. Packing, Scheduling oder Facility–Location an, die möglicherweise eine Anwendung der neuen Techniken erlauben. Ferner geben wir interessante offene Probleme an, die im Kontext dieser Arbeit auftreten.

Appendix

B Curriculum Vitae

Personal Data

Name	Gunnar Werner Klau
Date of Birth	September 18, 1970
Citizenship	German

Education

2000 – 2001	Technische Universität Wien, Austria PhD student, advised by Prof. Petra Mutzel
1997 – 2000	Max–Planck–Institut für Informatik, Saarbrücken, Germany PhD student, advised by Prof. Petra Mutzel
1994 – 1997	Universität des Saarlandes, Saarbrücken, Germany Diplom (≈ Master's) in computer science with minor in computational linguistics, thesis: *Quasi–Orthogonales Zeichnen planarer Graphen mit wenigen Knicken*, submitted to Prof. Kurt Mehlhorn.
1994	Università degli Studi di Padova, Italy Student of computer science, exchange programme
1991 – 1994	Bayerische Julius–Maximilians–Universität Würzburg, Germany Student of computer science with minor in linguistics. Vordiplom (≈ Bachelor's) in 1994.
1989 – 1991	Fern–Universität Hagen, Germany (distance teaching university) Auditor of economics during community service
1989	Claus–von–Stauffenberg–Schule, Rodgau, Germany German Abitur (≈ High school Diploma)

Work and Teaching Experience

2001	Four-months internship with Mitsubishi Electric Research Labs, Cambridge, Massachusetts, U.S.A. Human guided optimization for vehicle routing problems.
2000 – present	Technische Universität Wien, Austria University assistant. Teaching experience in numerous tutorials, practical courses, and lectures

1997 – 2000	Max–Planck–Institut für Informatik, Saarbrücken, Germany Assistant in the project *Automatical layout and labeling of state diagrams* in cooperation with Siemens AG, funded by the German Federal Ministry of Research. Teaching experience in numerous tutorials, seminars, and practical courses.
1996 - present	Developer of large parts of the C++ software library AGD
1995 - 1996	Part-time programmer for research projects at MPI Saarbrücken, implementation of graph drawing algorithms
1995	Six-week internship with the Faculty of Marine Technics, Klaipeda University, Lithuania (via IAESTE)
1992 - 1994	Part-time programmer for the Department of Political Sciences, Würzburg University, design and implementation of a software system to support systematical development of political theories
1991	Ten-week internship (mechanical engineering) with MTU München, Munich, Germany, now DaimlerChrysler Aerospace
1989 – 1991	Community Service (Zivildienst) 1990 – 1991: European Nature Heritage Fund, Radolfzell, Germany 1989 – 1990: Johanniter-Unfallhilfe, Rodgau, Germany
1989	Four-week internship (CAD) with ABB-Strömberg, Rödermark
1987 - 1995	Part-time programmer for consulting engineers office *ibas*, Rödermark, Germany, design and implementation of a software system for calculating power supplies of short-distance traffic systems used by AEG-Westinghouse, Frankfurt, Germany

Awards

1998	German Study Research Price (awarded for Master's thesis) Körber foundation, Hamburg, Germany

Publications

Chapters in Books

- *Automatic Layout and Labelling of State Diagrams*, with Petra Mutzel, to appear in *Mathematics—Key Technology for the Future*, Springer, 2002
- *Orthogonal Graph Drawing*, with Markus Eiglsperger and Sándor Fekete, in M. Kaufmann and D. Wagner, editors, *Graph Drawing: Methods and Models*, LNCS Tutorial 2025, pp. 140-193, Springer, 2001

Journal Articles

- *Optimal Labeling of Point Features in Rectangular Labeling Models*, with Petra Mutzel, Mathematical Programming, Series B, to appear 2002

Refereed Conference Proceedings

- *An Experimental Comparison of Orthogonal Compaction Algorithms*, with Karsten Klein and Petra Mutzel, in Proc. of the 8th Int. Symp. on Graph Drawing (GD 2000), LNCS 1984, Williamsburg, Virginia, USA, pp. 37-51, Springer, 2000

- *Optimal Labelling of Point Features in the Slider Model*, with Petra Mutzel, in Proc. of the 6th Ann. Int. Computing and Combinatorics Conf. (COCOON 2000), LNCS 1858, Sydney, Australia, pp. 340-350, Springer, 2000

- *Combining Graph Labeling and Compaction*, with Petra Mutzel, in Proc. of the 7th Int. Symp. on Graph Drawing (GD 1999), LNCS 1731, Štiřín Castle, Czech Republic, Springer, pp. 27-37, 1999

- *Optimal Compaction of Orthogonal Grid Drawings*, with Petra Mutzel, in Proc. of the 7th Int. Conf. in Integer Programming and Combinatorial Optimization (IPCO 99), LNCS 1610, Graz, Austria, pp. 304-319, Springer, 1999

- *AGD: A Library of Algorithms for Graph Drawing*, with Petra Mutzel, Carsten Gutwenger, Ralf Brockenauer, Sergej Fialko, Michael Krüger, Thomas Ziegler, Stefan Näher, David Alberts, Dirk Ambras, Gunter Koch, Michael Jünger, Christoph Buchheim and Sebastian Leipert, in Proc. of the 6th Int. Symp. on Graph Drawing (GD 98), LNCS 1547, Montréal, Canada, pp 456-457, Springer, 1998

Research Reports

- *Optimal Labelling of Point Features in Rectangular Labelling Models*, with Petra Mutzel, Report No. TR-186-1-00-04, Vienna University of Technology, Austria, 2000

- *An Experimental Comparison of Orthogonal Compaction Algorithms*, with Karsten Klein and Petra Mutzel, Report No. TR-186-1-00-03, Vienna University of Technology, Austria, 2000

- *Graph Drawing Algorithm Engineering with AGD*, with Carsten Gutwenger, Michael Jünger, Sebastian Leipert and Petra Mutzel, Report No. TR-186-1-00-02, Vienna University of Technology, Austria, 2000

- *Optimal Compaction of Orthogonal Grid Drawings*, with Petra Mutzel, Report No. MPI-I-98-1-031, Max Planck Institute of Computer Science, Saarbrücken, Germany, 1998

- *Quasi Orthogonal Drawing of Planar Graphs*, with Petra Mutzel, Report No. MPI-I-98-1-013, Max Planck Institute of Computer Science, Saarbrücken, Germany, 1998

Miscellaneous Publications

- *Visuelle Zeitenwende: Bilder - Technik - Reflexionen. Quasi-orthogonales Zeichnen*, contribution to the *German Study Research Price*, organized by the Körber foundation, awarded second, Leipzig, Germany, 1998, in German language
- *Quasi-orthogonales Zeichnen planarer Graphen mit wenigen Knicken*, diploma thesis at Saarland University, 1997, in German language

BIBLIOGRAPHY

P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, 11:209–218, 1998.

AGD. *Algorithms for Graph Drawing. User Manual*. Max-Planck-Institut für Informatik, Saarbrücken, Universität Halle, Universität zu Köln, Technische Universität Wien, 2001. URL `http://www.mpi-sb.mpg.de/AGD`.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

S. B. Akers, M. E. Geyer, and D. L. Roberts. IC mask layout with a single conductor layer. In *Proc. 7th Design Automation Workshop*, pages 7–16. ACM/IEEE, 1970.

D. Alberts, C. Gutwenger, P. Mutzel, and S. Näher. AGD-Library: A library of algorithms for graph drawing. In G. F. Italiano and S. Orlando, editors, *Proc. on the Workshop on Algorithm Engineering (WAE 1997)*, Venice, Italy, 1997. URL `http://www.dsi.unive.it/~wae97`.

M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Ann. Oper. Res.*, 16:201–240, 1988.

C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data-flow diagrams. *IEEE Transactions on Software Engineering*, SE-12(4):538–546, 1986.

R. Bellman. On a routing problem. *Quart. Appl. Math.*, 16:87–90, 1958.

P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Transactions on Computers*, 49(8):826–840, 2000.

S. Bhatt and S. Cosmadakis. The complexity of minimizing wire length in VLSI layouts. *Information Processing Letters*, 25:263–287, 1987.

F. J. Brandenburg. Nice drawings of graphs and trees are computationally hard. Technical Report MIP-8820, Universität Passau, Fakultät für Mathematik und Informatik, Passau, Germany, 1988.

S. Bridgeman, G. Di Battista, W. Didimo, G. Liotta, R. Tamassia, and L. Vismara. Turn-regularity and optimal area drawings of orthogonal representations. *Computational Geometry: Theory and Applications*, 16(1):53–93, 2000.

B. V. Cherkassky and A. V. Goldberg. Negative-cycle detection algorithms. *Math. Program.*, 85(2):277–311, 1999.

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*, 30:54–76, 1985.

J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Trans. Graph.*, 14(3):203–232, 1995.

S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158, 1971.

W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley & Sons, 1998.

CPLEX. *Reference Manual, Version 6.5*, 1999. ILOG Inc.

R. G. Cromley. A spatial allocation analysis of the point annotation problem. In *Proc. 2nd Internat. Symp. on Spatial Data Handling*, pages 38–49, 1986.

W. W.-M. Dai and E. S. Kuh. Global spacing of building-block layout. In C. H. Sequin, editor, VLSI '87, pages 193–205. Elsevier Science, 1987.

G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of prescribed size. In J. Kratochvíl, editor, *Proc. 8th Internat. Symp. on Graph Drawing (GD 1999)*, volume 1731 of *Lecture Notes in Computer Science*, pages 297–310, Štiřín Castle, Czech Republic, 1999a. Springer.

G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.

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

G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry: Theory and Applications*, 7:303–316, 1997.

R. Diestel. *Graph Theory*. Springer, Berlin, Germany, 1997.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

S. Doddi, M. V. Marathe, A. Mirzaian, B. M. E. Moret, and B. Zhu. Map labeling and its generalizations. In *Proc. of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA 1997)*, pages 148–157, New Orleans, LA, U.S.A., 1997.

J. Doenhardt and T. Lengauer. Algorithmic aspects of one-dimensional layout compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(5):863–879, 1987.

D. Dolev, F. T. Leighton, and H. Trickey. Planar embedding of planar graphs. *Advances in Computing Research*, 2, 1984.

D. Dolev and H. Trickey. On linear area embedding of planar graphs. Technical Report CS-81-876, Stanford University, U.S.A., 1981.

A. E. Dunlop. SLIP: Symbolic layout of integrated circuits with compaction. *Computer Aided Design*, 10(6):387–391, 1978.

P. Eades and P. Mutzel. Graph drawing algorithms. In M. Atallah, editor, *CRC Handbook of Algorithms and Theory of Computation*, pages 9/1–9/26. CRC Press, 1999.

J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

M. Eiglsperger, U. Fößmeier, and M. Kaufmann. Orthogonal graph drawing with constraints. In *10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pages 3–11, 1999.

J. Fanto. Postprocessing of GIOTTO drawings. Student project, Brown University, U.S.A., 1997.

L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, U.S.A., 1962.

M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Symp. Comput. Geom. (SoCG 1991)*, pages 281–288, 1991.

U. Fößmeier, C. Heß, and M. Kaufmann. On improving orthogonal drawings: The 4M-algorithm. In *Proc. of the 6th International Symposium on Graph Drawing (GD 1998)*, volume 1547 of *Lecture Notes in Computer Science*, pages 125–137, Montréal, Canada, 1998. Springer.

U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Proc. of the 3rd Int. Symp. on Graph Drawing (GD 1995)*, volume 1027 of *Lecture Notes in Computer Science*, pages 254–266, Passau, Germany, 1996. Springer.

U. Fößmeier and M. Kaufmann. Algorithms and area bounds for nonplanar orthogonal drawings. In G. Di Battista, editor, *Proc. of the 5th Int. Symp. on Graph Drawing (GD 1997)*, volume 1353 of *Lecture Notes in Computer Science*, pages 134–145, Rome, Italy, 1997. Springer.

E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, U.S.A., 1979.

M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified *NP*-complete graph problems. *Theor. Comput. Sci.*, 1:237–267, 1976.

A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. In *Proc. of the DIMACS Int. Workshop on Graph Drawing (GD 1994)*, volume 894 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 1994.

A. Garg and R. Tamassia. A new minimum cost flow algorithm with applications to graph drawing. In S. North, editor, *Proc. 5th Internat. Symp. on Graph Drawing (GD 1996)*, volume 1190. Springer, 1997.

R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, pages 275–278, 1958.

A. Gregori. Unit length embeddings of binary trees on a square grid. *Information Processing Letters*, 31:167–172, 1989.

M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 2 edition, 1994.

C. Gutwenger. Design und Implementierung einer Algorithmen–Bibliothek zum Zeichnen von Graphen. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 1999. In German language.

C. Gutwenger, M. Jünger, G. W. Klau, S. Leipert, and P. Mutzel. Graph drawing algorithm engineering with AGD. Technical Report TR-186-1-00-04, Technische Universität Wien, Institut für Computergraphik und Algorithmen, Vienna, Austria, 2000.

C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. In *Proc. of the 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA 2001)*, pages 246–255. ACM-SIAM, 2001.

S. A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

F. Hoffmann and K. Kriegel. Embedding rectilinear graphs in linear time. *Information Processing Letters*, 29(2):75–79, 1988.

M.-Y. Hsueh. Symbolic layout and compaction of integrated circuits. Technical Report UCB/ERL M79/80, Electronic Research Laboratory, University of California, Berkeley, CA, U.S.A., 1979.

E. Imhof. Die Anordnung der Namen in der Karte. *International Yearbook of Cartography*, 2:93–129, 1962.

C. Iturriaga and A. Lubiw. *NP*-hardness of some map labeling problems. Technical Report CS-97-18, University of Waterloo, Canada, 1997a.

C. Iturriaga and A. Lubiw. Elastic labels: The two-axis case. In G. Di Battista, editor, *Proc. 6th Internat. Symp. on Graph Drawing (GD 1997)*, volume 1353 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 1997b.

M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings. *Algorithmica, Special Issue on Graph Drawing*, 16(1):33–59, 1996.

M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In W. Cook, L. Lovász, and P. Seymour, editors, *Combinatorial Optimization: Papers from the DIMACS Special Year*, number 20 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 111–152. AMS, Providence, RI, U.S.A., 1995.

M. Jünger and S. Thienel. The ABACUS system for branch and cut and price algorithms in integer programming and combinatorial optimization. *Softw. Pract. Exper.*, 30: 1325–1352, 2000.

K. G. Kakoulis and I. G. Tollis. An algorithm for labeling edges of hierarchical drawings. In G. Di Battista, editor, *Proc. of the 5th Int. Symp. on Graph Drawing (GD 1997)*, volume 1353 of *Lecture Notes in Computer Science*, pages 169–180, Rome, Italy, 1997a. Springer.

K. G. Kakoulis and I. G. Tollis. On the edge label placement problem. In *Proc. 5th Internat. Symp. on Graph Drawing (GD 1996)*, volume 1190 of *Lecture Notes in Computer Science*, pages 241–256. Springer, 1997b.

R. M. Karp and C. H. Papadimitriou. On linear characterizations of combinatorial optimization problems. *SIAM J. Comput.*, 11:620–632, 1982.

T. Kato and H. Imai. The *NP*-completeness of the character placement problem of 2 or 3 degrees of freedom. In *Record of Joint Conf. of Electrical and Electronic Engineers*, page 1138, Kyushu, Japan, 1988.

M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*. Springer, 2001.

G. Kedem and H. Watanabe. Graph optimization techniques for IC–layout and compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-3(1):12–20, 1984.

G. W. Klau. A segment-saving variant of Tamassia's bend minimization algorithm. Manuscript, 2001.

G. W. Klau and P. Mutzel. Quasi-orthogonal drawing of planar graphs. Technical Report MPI-I-98-1-013, Max–Planck–Institut für Informatik, Saarbrücken, Germany, 1998.

K. Klein. Flußbasierte orthogonale Zeichenverfahren für Graphen mit variablen Knotengrößen. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 2000. In German language.

D. E. Knuth. *The Art of Computer Programming*, volume 1: Fundamental Algorithms. Addison-Wesley, Reading, MA, U.S.A., 1968.

L. Kučera, K. Mehlhorn, B. Preis, and E. Schwarzenecker. Exact algorithms for a geometric packing problem. In *Proc. 10th Sympos. Theoret. Aspects Comput. Sci. (STACS 1993)*, volume 665 of *LNCS*, pages 317–322. Springer, 1993.

A. S. LaPaugh. VLSI layout algorithms. In M. J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, pages 23/1–23/26. CRC Press, 1998.

T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, New York, 1990.

T. Lengauer and K. Mehlhorn. VLSI complexity, efficient VLSI algorithms and the HILL design system. In C. Trullemans, editor, *Algorithmics for* VLSI, pages 33–89. Academic Press, New York, U.S.A., 1986.

P. C. Liu and R. C. Geldmacher. On the deletion of nonplanar edges from a graph. In *Proc. 10th S-E Conf. on Combionatorics, Graph Theory and Computation*, pages 727–738, Boca Raton, FL, U.S.A., 1977.

F. M. Maley. An observation concerning constraint-based compaction. *Information Processing Letters*, 25(2):119–122, 1987.

J. Marks and S. Shieber. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard University, Cambridge, MA, U.S.A., 1991.

K. Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. Springer, 1984.

K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996.

K. Mehlhorn and S. Näher. LEDA. *A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 1999.

E. F. Moore. The shortest path through a maze. In *Proc. of the Int. Symp. on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

K. Mühlenbacher. Heuristiken zur Kompaktierung orthogonaler Zeichnungen. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 2001. In German language. In Preparation.

P. Mutzel. *The Maximum Planar Subgraph Problem*. PhD thesis, Institut für Informatik, Universität zu Köln, Germany, 1994.

M. W. Padberg and M. Rao. The Russian method for linear inequalities III: Bounded integer programming. GBA Working Paper 81-39, 1981.

M. Patrignani. On the complexity of orthogonal compaction. In F. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *Proc. 6th International Workshop on Algorithms and Data Structures (WADS 1999)*, volume 1663 of *Lecture Notes in Computer Science*, pages 56–61. Springer, 1999.

M. Schlag, Y.-Z. Liao, and C. K. Wong. An algorithm for optimal two-dimensional compaction of VLSI layouts. *Integration, the VLSI Journal*, 1:179–209, 1983.

M. Schlag, F. Luccio, P. Maestrini, D. T. Lee, and C. K. Wong. A visibility problem in VLSI layout compaction. *Advances in Computing Research*, 2:259–282, 1984.

A. Schrijver. *Theory of Integer and Linear Programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1986.

J. M. Six, K. G. Kakoulis, and I. G. Tollis. Techniques for the refinement of orthogonal graph drawings. *Journal of Graph Algorithms and Applications*, 4(3):75–103, 2000.

J. A. Storer. On minimal node-cost planar embeddings. *Networks*, 14:181–212, 1984.

T. Strijk and M. van Kreveld. Practical extensions of point labeling in the slider model. In *Proc. 7th ACM Symp. Adv. Geogr. Inform. Syst.*, pages 47–52, 1999.

K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981.

R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988.

R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Transactions on Circuits and Systems*, 36(9):1230–1234, 1989.

S. Thienel. ABACUS—*A Branch-And-CUt-System*. PhD thesis, Universität zu Köln, Germany, 1995.

P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.

M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13:21–47, 1999.

B. Verweij. *Selected Applications of Integer Programming: A Computational Study*. PhD thesis, Universiteit Utrecht, The Netherlands, 2000.

B. Verweij and K. Aardal. An optimisation algorithm for maximum independent set with applications in map labelling. In *Proc. 7th Annu. Europ. Symp. Algorithms (ESA 1999)*, volume 1643 of *Lecture Notes in Computer Science*, pages 426–437, Prague, Czech Republic, 1999. Springer.

G. Vijayan and A. Wigderson. Rectilinear graphs and their embeddings. *SIAM J. Comput.*, 14(2):355–372, 1985.

H. Watanabe. IC *Layout Generation and Compaction Using Mathematical Optimization*. PhD thesis, University of Rochester, NY, U.S.A., 1984.

R. Weiskircher. *New Applications of* SPQR*–Trees in Graph Drawing*. PhD thesis, Technische Fakultät I der Universität des Saarlandes, Saarbrücken, Germany, 2002. In preparation.

J. D. Williams. STICKS—a graphical compiler for high-level LSI design. In *Proc. AFIPS Conf.*, volume 47, pages 289–295, 1978.

G. J. Woeginger, 2000. Institut für Mathematik, TU Graz, Austria. Personal communication.

A. Wolff and T. Strijk. The map labeling bibliography, 2001. URL `http://www.math-inf.uni-greifswald.de/map-labeling/bibliography`.

P. Yoeli. The logic of automated map lettering. *The Cartographic Journal*, 9:99–108, 1972.

T. Ziegler. *Crossing Minimization in Automatic Graph Drawing*. PhD thesis, Technische Fakultät I der Universität des Saarlandes, Saarbrücken, Germany, 2001.

S. Zoraster. The solution of large 0-1 integer programming problems encountered in automated cartography. *Oper. Res.*, 38(5):752–759, 1990.

INDEX