TU WIEN Informatics

Grouping and Ordering Constraints in Boundary Labeling

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Thomas Depian, BSc
Matrikelnummer 11807882

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Mitwirkung: Projektass. Dipl.-Ing. Soeren Terziadis
Projektass. Dipl.-Ing. Markus Wallinger

Wien, 16. November 2023

Thomas Depian	Martin Nöllenburg

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Grouping and Ordering Constraints in Boundary Labeling

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Thomas Depian, BSc
Registration Number 11807882

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Assistance: Projektass. Dipl.-Ing. Soeren Terziadis
Projektass. Dipl.-Ing. Markus Wallinger

Vienna, 16th November, 2023

Thomas Depian | Martin Nöllenburg

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Thomas Depian, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 16. November 2023

Thomas Depian

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to thank Soeren Terziadis, Markus Wallinger, and Martin Nöllenburg for guiding me through the jungle of writing a thesis. In particular, I want to thank Martin Nöllenburg for showing me the world of research and giving me the opportunity to work with him and his colleagues on challenging but exciting projects. Thinking back two years, I never imagined that I would be presenting some of our results at workshops and conferences (or writing my thesis while waiting at the airport for my flight to one of them), and I look forward to finding answers to further open research questions together. I also thank Benjamin Niedermann for providing me with sample instances from the Sobotta atlas of human anatomy and Melanie Kirchgessner and the team behind ScienceFacts.Net for permitting me to use their images in this thesis.

Even if it is not written on the cover page, this work bears the signature of my friends who have always supported me on my journey even if I disappeared behind work. In particular, I want to thank Andreas and Anton, who made Vienna a second home for me, and Christoph and Michael, who have accompanied me on so many (university) projects and kept things fun. Thanks must also go to my girlfriend Johanna for being my motivator, travel guide, and partner in crime.

I cannot end this section before thanking my parents. Without them, I would not be where I am now, and I am deeply grateful to them for always encouraging me to go my own way, even if that meant coming home less often.

This thesis is for you.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Illustrationen beinhalten oft Merkmale mit einem Label für deren Name oder Beschreibung. Um zu verhindern, dass wichtige Teile der Illustration mit einem Label verdeckt werden und trotzdem ansprechende und verständliche Abbildungen zu erstellen, können wir die Labels um die Illustration herum platzieren und kreuzungsfreie Führungslinien verwenden, um eine eindeutige Zuordnung zwischen dem Label und dem Merkmal sicherzustellen. Vor allem in medizinischen und technischen Zeichnungen wenden wir häufig diese Methode der externen Beschriftung an. Jedoch gibt es in diesen Gebieten auch weitere Beschränkungen, welche aus der Semantik, die die Merkmale tragen, hervorgehen. Ein Beispiel hierfür ist eine Gruppe von Merkmalen, die zusammen ein größeres Objekt darstellen und daher nebeneinander beschriftet werden sollen. Obwohl die Literatur eine Vielzahl von Ansätzen zur Erstellung von externen Beschriftungen beschreibt, berücksichtigt kaum eine davon diese semantischen Beschränkungen.

In dieser Masterarbeit betrachten wir daher nochmal Randbeschriftungen, eine Unterart von externen Beschriftungen, bei der die Labels entlang eines rechteckigen Randes um die Illustration herum platziert werden. Wir erweitern diese um *Gruppierungs-* und *Ordnungsbeschränkungen*, welche die Menge der gültigen Beschriftungen einschränken. Gruppierungsbeschränkungen verlangen, dass ein Teil der Labels nacheinander auf dem Rand platziert wird. Ordnungsbeschränkungen definieren eine partielle Ordnung auf den Merkmalen. Um diese Beschränkungen darzustellen, schlagen wir PQ-A-Graphen vor, eine Erweiterung von PQ-Bäumen. Durch die Interpretation einer Beschriftung als eine Permutation der Merkmale, lässt sich zeigen, dass wir alle relevanten Beschränkungen als PQ-A-Graphen darstellen können. Wir präsentieren einen Algorithmus, um einseitige Randbeschriftungen mit achsenparallelen Führungslinien, welche höchstens eine Biegung aufweisen, zu erstellen. Mithilfe von echten und künstlichen Instanzen evaluieren wir die Performance des Algorithmus und analysieren die erzeugten Beschriftungen. Des Weiteren werden mögliche Interpretationen unserer Beschränkungen in mehrseitigen Randbeschriftungen diskutiert und wir zeigen, dass das Finden einer gültigen Randbeschriftung in unserem Interpretationsvorschlag bereits für zwei Seiten NP-vollständig ist. Abschließend schlagen wir Variationen unseres Problems vor, welche wir als interessant und relevant für praktische Anwendungen erachten, und präsentieren, als Anstoß für weitere Forschung, hierfür erste Ergebnisse.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Illustrations often include features with a label for their name or description. To avoid obstructing crucial parts of the illustration with a label and still create appealing and comprehensible figures, we can place the labels around the illustration and use non-crossing leader lines to ensure that the connection between a label and the feature is unambiguous. Especially in medical and technical drawings, we frequently apply this external labeling technique. However, in these domains, we can encounter additional constraints arising from the semantics of the features, such as groups of features that constitute parts of a larger object and should be labeled next to each other. Despite the fact that the literature proposes a multitude of approaches to compute external labelings, hardly any of them take semantic constraints into account.

In this thesis, we revisit boundary labeling, a branch of external labeling, where all labels are placed along a rectangular boundary around the illustration. We enrich it by *grouping* and *ordering constraints* that restrict the set of feasible labelings. Grouping constraints enforce that a subset of the labels must appear consecutively on the boundary while ordering constraints define a partial order on the features. We propose PQ-A-Graphs, an extension of PQ-Trees, to represent our constraints. By interpreting a labeling as a permutation of the features, we are able to show that all relevant grouping and ordering constraints can be encoded as PQ-A-Graphs. We present an algorithm to compute a one-sided boundary labeling with axis-aligned leader lines that have at most one bend. Using real-world and artificial instances, we evaluate the performance of the algorithm and analyze the resulting labelings. Furthermore, we discuss an interpretation of our constraints in multi-sided boundary labelings and show that finding a feasible labeling in the suggested interpretation is NP-complete, even if we only consider two sides. Finally, we propose variations of our problem that we deem interesting and relevant for practical applications and provide preliminary results as starting points for future research.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

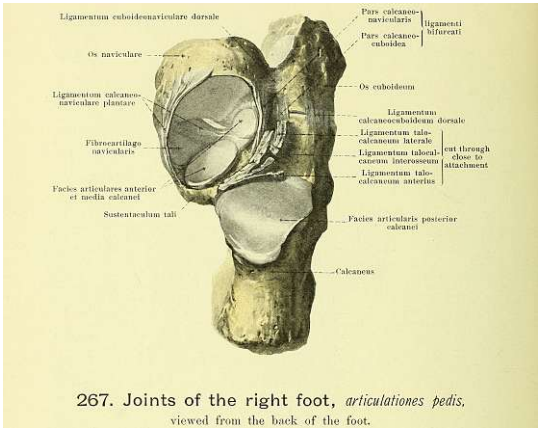
Introduction

"As he sketches the thing out on paper or he studies an illustration made by someone else, he is helped in gaining a mental image of the subject. But, more than this, illustrations are hard taskmasters. They force us to think clearly and logically. One can write around a subject of which one is not quite sure. But, it is hard to leave holes or blank spaces in a picture."

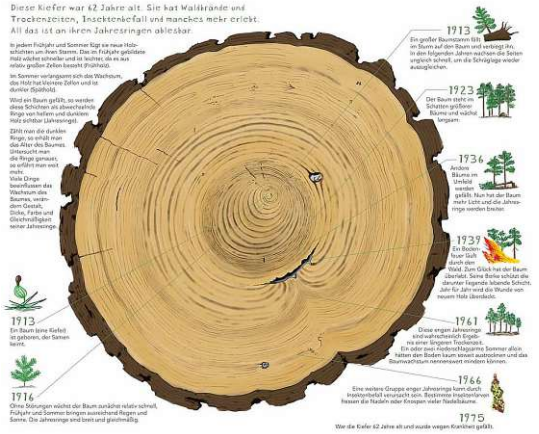
— Frank H. Netter, M.D. (1906 – 1991) [Net81, p. 226]

We encounter labeled illustrations of components and concepts of our world on a daily basis. While most people think of cartographic applications, such as *Google Maps*, these are only a fraction of what we summarize under the term *labeling*. Labelings also appear outside geographic information systems, and representatives of those have been with us since our childhood. Starting at young ages, when we label objects in picture books to introduce vehicles and animals to our children, they quickly become indispensable for many pupils and students. This is no surprise, as Mayer has shown that accompanying labeled illustrations make it easier to understand textual descriptions [May89]. Yet, blindly adding figures to text without considering the need and, in particular, the prior knowledge of the readers is being critically questioned, and illustrations that do not contain (all relevant) labels seem to have little to no effect [MG90]. Therefore, guidelines have emerged that aid in developing targeted and useful labeled illustrations.

In her guidelines for scientific and medical illustrations, Briscoe pointed out that a good figure has "concise and consistent labels" [Bri90, p. 3]. Furthermore, we should "not obscure important details with labels" [Bri90, p. 35]. Medical illustrations often contain many, sometimes even fine-grained, important details. To adhere to these guidelines, designers tend to place the labels outside the illustrations, thus creating an *external labeling*. They use polyline *leaders* to connect labels with the feature points, here called *sites*, they describe. Already medical illustrations from the early 20th century, as the one by Spalteholz in Figure 1.1a, use this technique. In his figure, Spalteholz arranges



(a) An illustration of the right foot from the *Hand Atlas of Human Anatomy* by Werner Spalteholz, 1906 [Spa06, p. 222].



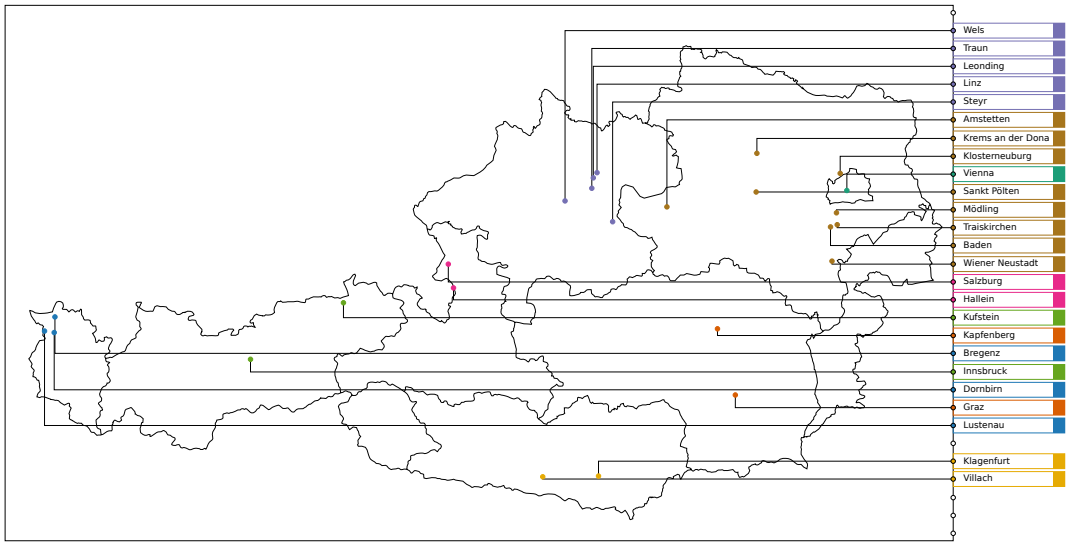
(b) Illustration on a board of the *Waldlehrpfad im Fürther Stadtpark*, a forest natural trail in Germany. © Melanie Kirchgessner, 2019 [Kir19].

Figure 1.1: Existing illustrations that contain (a) grouping and (b) ordering constraints.

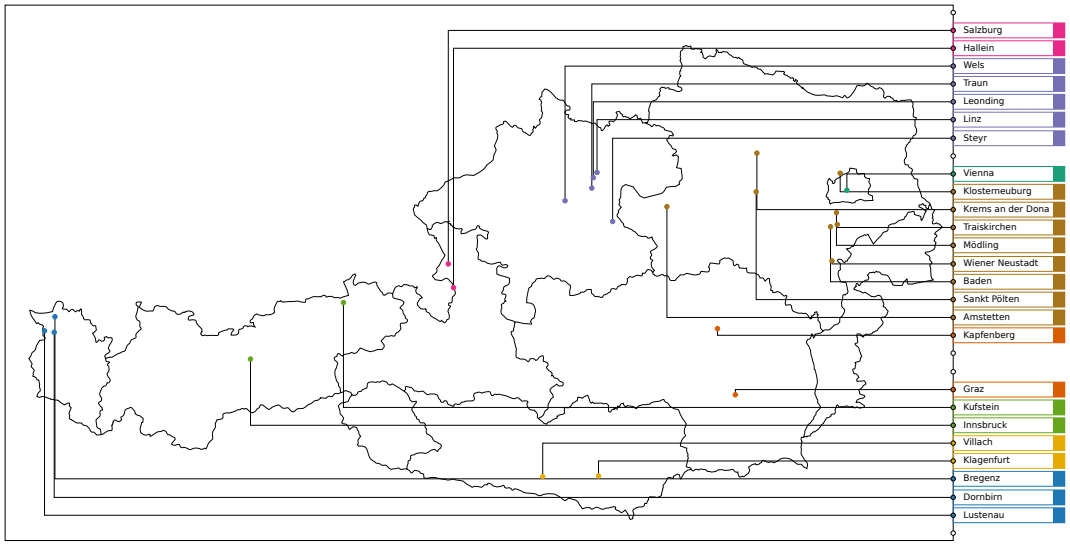
the labels around the illustration in a readable way and uses straight-line non-crossing leaders to connect sites with labels. Furthermore, he ensures that the leaders are not too long, thus following Tufte's principle of maximizing the *data-ink ratio* [Tuf01].

Knowing that designers of similar drawings stated that they need about two hours to accommodate illustrations and their textual information on a double page of a book [NNR17], one can imagine how time-consuming the creation of an atlas of human anatomy with its hundreds of illustrations is. Therefore, it is no wonder that over the years, the literature has proposed various approaches to at least automatically place labels around an illustration and connect them to sites with leaders in an ink-minimal way, i.e., using short leaders. These approaches sometimes create external labelings in the style of Figure 1.1a, but another common labeling style is *boundary labeling*, where we place the labels on a(n invisible) rectangular boundary around the illustration. In an attempt to categorize and compare the plethora of results, Bekos et al. [BNN21] identified ten open challenges that should be tackled in this area both by the algorithms and the visual computing community. The fourth challenge states that we should "[d]evelop more generic, less problem-specific heuristics and algorithms for external labeling" [BNN21, p. 97]. Taking a closer look at Figure 1.1a, we can observe that Spalteholz sometimes grouped smaller parts into bigger ones by utilizing a curly bracket. Niedermann et al. [NNR17] even stated that in a sample of 202 figures from the Sobotta atlas of human anatomy [WP13], they could identify 18 with such a curly bracket. Hence, in this atlas, almost every tenth figure contains an explicit group of labels that belong semantically together. Of course, this requires that the labels for the group are next to each other, i.e., not separated by a label that is not part of the group. We can also find examples outside the medical domain where we should group semantically related labels, for example,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a)



(b)

Figure 1.2: Sample labelings of the 25 largest Austrian cities generated by our implementation that we describe in Chapters 4 and 5. Cities in the same state have the same color. Labeling (a) does not respect the grouping constraints, whereas Labeling (b) does.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

in map labeling. Consider Figures 1.2a and 1.2b, which show two labelings of the 25 largest cities of Austria. We created them with our algorithm, which we will describe in greater detail in this thesis. Figure 1.2a shows the leader-length-minimal labeling with axis-parallel leaders. Given that Austria consists of nine states, it would be natural to have the labels for the cities of the same state next to each other. The colors in Figure 1.2 indicate cities that are in the same state, and we can see that in Figure 1.2a several of these groups do not occupy continuous regions on the boundary of the figure. For example, the two *Styrian* cities *Kapfenberg* and *Graz*, marked orange, are separated by cities in *Tyrol* and *Vorarlberg*. Considering a labeling that respects these groups, as the one in Figure 1.2b, we might observe longer leaders but arguably also a more consistent appearance. Furthermore, respecting the groups allows an illustrator in a post-processing step to place information next to the groups, such as the name of the state. Using a curly bracket, as in Figure 1.1a, these groups can also be made explicit.

Although grouping constraints are arguably the most important manifestation of semantic constraints, they are not the only ones. In existing illustrations, we can sometimes find an order on the sites that is reflected in the order of the labels. Consider, for example, Figure 1.1b, which describes the events that we can read off the annual rings of a tree. On both sides of the illustration, the rings are labeled from the inside out, thus following the chronological order of the years, even if this implies that the leaders are longer than needed. Total orders, such as chronological orders, allow only one feasible ordering, thus mostly dictating the labeling. However, if we order the rings by the decade, then this is no longer the case: For two rings, we can only unambiguously determine their order if they are from different decades. Furthermore, nothing hinders us from combining semantic constraints of different types. For example, we could order the different layers of the sun from inside out but also partition the layers into *inner* and *outer layers*, as in Figure 1.3. Note that due to the grouping of the labels for the layers, the *sun spots* are labeled below the illustration, which increases the length of the leader for this label.

Including grouping and ordering constraints comes at a(n algorithmic) cost. Although such constraints are arguably common in real-world examples, they have not been extensively considered yet, and especially ordering constraints lack algorithmic support, as we will see in Section 1.1. Our aim with this thesis is to change that and investigate how we can generate boundary labelings of point data in the presence of grouping and ordering constraints. We will incorporate them into the existing model for boundary labeling and extend common approaches accordingly. In particular, we will take a closer look at their impact on the running time of these algorithms and the computational complexity of the boundary labeling problem in the presence of such constraints. While we see our main contribution in these theoretic results, we will also consider the performance of our algorithms from a practical perspective and analyze how such constraints affect the quality of the generated labelings. We believe that this could help overcome the challenge of the lack of generic algorithms identified by Bekos et al. [BNN21].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

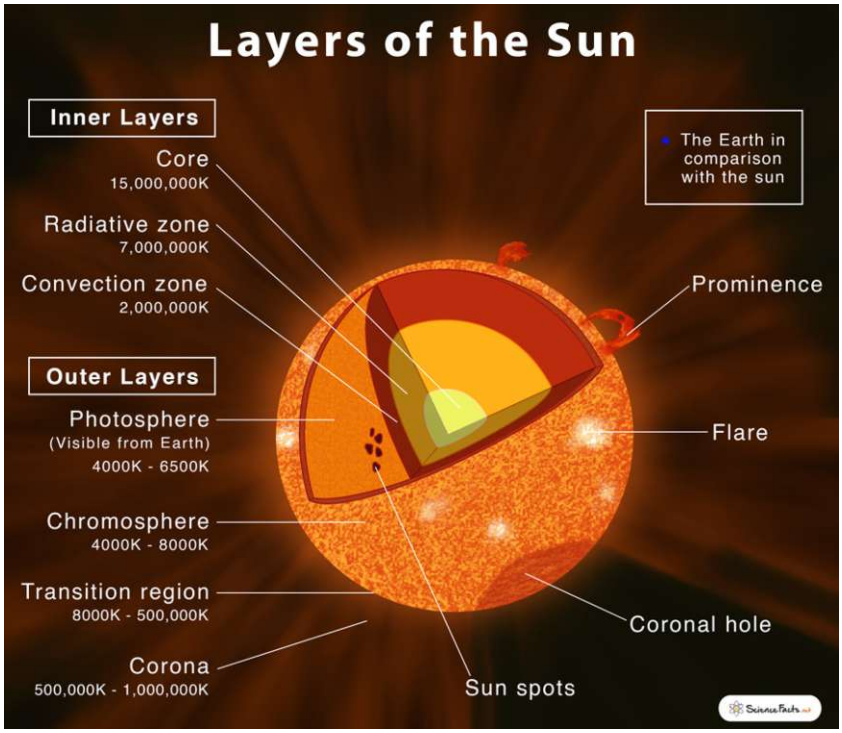


Figure 1.3: Illustration of the layers of the sun. © ScienceFacts.net, 2023 [BMs23].

1.1 Related Work

Algorithms for enriching illustrations with textual or graphical descriptions of *feature objects* have a long-standing tradition in the literature. The roots date back to the 1970s when Yoeli [Yoe72] and Imhof [Imh75] described basic principles for map labeling that are valid until today. To the best of our knowledge, *map labeling* is one of the first areas of computational geometry that deals with labeling feature objects, commonly point features. Formann and Wagner [FW91] created a foundation on which many papers still build today. On the one hand, they proposed the simple yet versatile *fixed position model* that assigns each feature point a constant number of candidate positions on which we can place a *label*. On the other hand, they showed that already allowing only four candidate positions makes finding an overlap-free labeling for a set of points NP-complete, even for uniform square labels [FW91].

While the spatial proximity between the feature point and its label is advantageous, such *internal labelings* have the downside that the labels usually occlude large, and probably also important, parts of the underlying illustration. To circumvent this limitation, Fekete and Plaisant [FP99] proposed the *excentric labeling* technique, where they dynamically label the feature points in a circular neighborhood around the mouse pointer. Their work already contains the main ingredients we nowadays use in *external labeling*: They place *rectangular labels* outside the illustration in a non-overlapping way, thus not hiding any

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

information, and connect the *ports* $\mathcal{P}$ of labels with their feature points, so-called *sites* $\mathcal{S}$, using (non-crossing) *leaders*, to avoid ambiguities as much as possible. Since then, many papers have proposed approaches to create external labelings for point features in static [BHKN09; NNR17] and interactive [FHS+12; GBNH21; GHS06a] settings, where in the latter, the labelings have to be updated upon user interaction. However, approaches where we place some labels outside the illustration and the remaining inside, thus creating a *mixed labeling*, have also been considered [BKPS11; CPWN22; LNS16].

As we can find external labelings in atlases of human anatomy or technical drawings, the proposed algorithms should compute a labeling that adheres to defined style guides and optimizes desired quality criteria. To incorporate these guidelines and criteria, previous work defined the set of feasible labelings based on them by, for example, enforcing crossing-free leaders [BKSW07] or providing a set of candidate label positions [BCK+18], modeled them in the optimization function [NNR17], or tried to learn them from existing handmade drawings [VVAH07]. The latter approach has recently also gained attention for creating internal labelings [BČČ23].

In the approach by Niedermann et al. [NNR17], the authors formalized guidelines commonly used by illustrators of atlases of human anatomy. They enforced a so-called *staircase labeling*, which we can summarize as being able to extend the width of the labels arbitrarily in one direction without introducing overlapping labels. Based on these guidelines, they proposed an $\mathcal{O}(|\mathcal{S}|^4|\mathcal{P}|^4)$-time dynamic programming algorithm to place labels for a set of sites along a convex contour that mimics the shape of the underlying illustration and connect the labels with non-crossing straight-line leaders to their sites. Recall that $\mathcal{S}$ and $\mathcal{P}$ denote the set of sites and (candidate) ports, respectively. The authors continuously sub-divided the contour until they were left with trivial instances. With their approach, they can label around 200 illustrations in thirty minutes, the time an illustrator usually needs to label a single illustration [NNR17]. Furthermore, their approach is versatile by allowing them to incorporate other desiderata as hard or soft constraints. This includes enforcing that a group of labels must appear consecutively on the contour and thus gives a first algorithm that supports grouping constraints. However, since the authors considered this only a potential extension of their algorithm, it was not created with such constraints in mind. Furthermore, they did not analyze the impact such constraints have on the running time and space consumption of the algorithm.

Enforcing that the labels are placed on a rectangular boundary around the figure might be seen as a restriction. Yet, *boundary labeling* has gained much attention in the literature. Its origins can be found in the work by Bekos et al. [BKSW07]. They proposed labelings where we place the labels on one or several sides of a rectangular boundary that encloses the sites. Furthermore, they considered straight-line leaders, which they called *s-leaders*, and poly-line leaders, consisting of several parallel and orthogonal segments, so-called *(o)po-leaders*. They used various techniques, such as dynamic programming, sweep-lines, bipartite matchings in the Euclidean space, and geometric arguments to reroute leaders and remove crossings. With them, they obtained in polynomial time leader-length-minimal labelings in various settings. Furthermore, they observed that we can reduce

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Partition to the problem of placing non-uniform height labels on two opposite sides of the boundary, showing that the corresponding labeling problem is weakly NP-complete. Thus, we often assume that we work with uniform-height labels. However, apart from the work by Bekos et al., other papers also contain NP-hardness results for non-uniform height labels under different settings [BKNS10; FS16].

There exists a rich body of literature that builds on or extends these initial results. While many of them assume rectilinear or straight-line leaders, Bekos et al. [BKNS10] proposed new leader styles, including *do*-leaders. They consist of two line segments, where one runs diagonally towards the label. The authors showed that we can find a leader-length-minimal one-sided labeling with *do*-leaders in $\mathcal{O}(|\mathcal{S}|^3)$ time. Given the different leader styles, it is natural to raise the question of which of them allows for the most readable figure. Barth et al. [BGNN19] compared the readability of *s*-, *po*-, *opo*, and *do*-leaders in a user study. They drew the conclusion that there is not much difference in the error rates between *s*-, *po*-, and *do*-leaders. However, the authors could observe faster response times for *po*-leaders, and the participants expressed their preference for *do*-leaders.

Benkert et al. [BHKN09] proposed a dynamic programming algorithm to produce one- and two-sided boundary labelings for *po*- and *do*-leaders for uniform height labels in $\mathcal{O}(|\mathcal{S}|^3)$ and $\mathcal{O}(|\mathcal{S}|^5)$ time, respectively. Their algorithm works for any optimization function that evaluates the quality of a single leader independent of the remaining labeling. However, this does not include grouping or ordering constraints, as they have to assess the placement of multiple leaders at a time. Nevertheless, the task of leader-bend-minimization falls under this category. Benkert et al., therefore, solved the open problem of Bekos et al. [BKSW07] of obtaining bend-minimal labelings for *po*-leaders. Similar to Niedermann et al. [NNR17], they combined labelings for smaller instances into a labeling for a larger one by using the leaders for sites as dividers for the instance. This will also appear in our dynamic programming algorithm that we will present in Chapter 4.

Multi-sided boundary labeling problems have also received attention in the literature. We can extend some of the results for the one-sided setting to two sides, most of the time by adapting the signature of the dynamic programming table, thus increasing the runtime significantly [BHKN09; FS16]. Other results are motivated by some specific application, such as placing comments in text documents [KLW14]. Note that these results assume that we put the labels on two opposite sides of the boundary, usually the left and right sides. However, there are also results for placing them on two adjacent sides or even three or four sides of the boundary [GAH23; KNR+16]. To some extent related to multi-sided problems are variants where we allow putting the labels in multiple rows of the same side or temporarily route leaders in one-sided settings at the opposite side of the boundary. The literature contains polynomial-time algorithms and computational complexity results for the former [GHN15] and the latter [LPT+11] variants.

Many-to-one labelings can be seen as applying a particular form of grouping constraints, where several sites describe the same feature and thus can be labeled by a single label connected to the sites via multiple leaders that emerge from a common *backbone*. If

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

we label each such set of sites with a single label, we can model this as a grouping constraint where all labels but one have zero height. Bekos et al. [BCF+15] analyzed different variations of this model where we minimize the number of labels, thus rewarding if multiple sites are connected to a single label, minimize the leader length, or minimize the number of crossings if we allow leaders to cross.

Due to the unit height assumption of the labels, many papers on boundary labeling implicitly assume that the height of the labels is maximal. Thus, a labeling has no (or a uniform-sized) gap between two consecutive labels on the boundary. Nöllenburg et al. [NPS10] relaxed this assumption and allowed irregular gaps between labels. If multiple labels touch each other, they form a so-called *cluster*. The authors observed that in a leader-length-minimal one-sided boundary labeling with *po*-leaders, we can assume that each cluster contains a direct leader, i.e., a *po*-leader that runs only horizontally, connecting to the median label, after possibly introducing a dummy-label of zero height in case there is no (unique) median label. This observation was used to propose a polynomial-time algorithm. Huang et al. [HPL14] extended these results for different combinations of uniform and non-uniform height labels, *po*- and *opo*-leaders, leader-length- and -bend-minimization, and one- and two-sided settings.

One motivation for external labeling is to occlude the underlying illustration as little as possible. Especially in medical drawings we often have to deal with important parts that not only should not be hidden by labels but must also not be crossed by leaders. To fulfill this requirement, Fink and Suri [FS16] proposed an algorithm where we have also given a set of rectangular objects that should not be crossed by the leaders. These objects serve as *obstacles* that we can use to represent crucial parts of the illustration that should not be distracted. Fink and Suri noted that in the presence of obstacles, it is not sensible to consider an a priori given set of potential label positions on the boundary, as this is likely to lead to infeasible instances. Rather, they let the labels slide along the boundary in the spirit of the papers on clustering the labels [NPS10; HPL14]. However, they observed that for a leader-length-minimal labeling, it is sufficient to consider $\mathcal{O}(c^2)$ candidate ports on the boundary induced by the positions of the sites and obstacles, where c denotes the complexity of the obstacles and the number of sites. Thus, they could use the well-known dynamic programming algorithm that uses leaders to decompose an instance. Fink and Suri not only proposed polynomial-time algorithms for various leader styles in the one- and two-sided setting but also showed by a reduction from Partition that finding a crossing-free labeling with non-uniform height labels is NP-complete even when we can only label on one side of the boundary. It will turn out that we can adapt their reduction to also work in the presence of grouping or ordering constraints instead of obstacles.

Kaufmann [Kau09] published a survey in 2009 on results concerning boundary labeling. Later, in 2021, Bekos et al. [BNN21] published a book containing a more extensive survey on external labeling. One of the goals of the latter survey is to put forward a common nomenclature in the field of external labeling and categorize the different results depending on their features. This categorization makes the lack of support for semantic constraints, such as grouping and ordering constraints, in external labeling and thus also

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

in boundary labeling clear. To the best of our knowledge, no paper explicitly incorporates ordering constraints. For grouping constraints, the survey reports a handful of papers that support the grouping or clustering of labels. However, a more detailed analysis of the mentioned papers and their respective related literature makes clear that only a few of them conceptualize grouping constraints in the way we do in this thesis. Some approaches placed collections of (similar) labels on top of each other and only showed the top-most label [GBNH21; FHS+12]. Also, the already mentioned papers on placing the labels in several stacks with irregular gaps can be seen as a manifestation of clustering the labels. Thus, many see clusters as a set of labels that happen to be close to each other without any other semantic association. Furthermore, some approaches try to place (the labels in) such clusters in a visually appealing way, for example, aligned and next to each other [GHS06a; VVAH07].

To the best of our knowledge, apart from Niedermann et al. [NNR17], only the papers by Götzelmann et al. [GHS06b] and Gedicke et al. [GAH23] support the grouping of labels in the same way as we intend to do in this thesis. Götzelmann et al. [GHS06b] assumed as additional input an ontology[1] on the sites. While we can form groups using an ontology, they used heuristic label placements and neglected crossings among leaders from sites of different groups. This is in contrast to our setting, where we aim for a labeling where no two leaders cross, independent of whether the corresponding sites are in the same group. On top of that, they labeled 3D visualizations at interactive speed and thus did not give a guarantee on the quality of the produced labelings. The paper by Gedicke et al. [GAH23] was published while we worked on the results for this thesis. They proposed an approach to create so-called *situation maps* used by emergency services to assess the situation and plan the operation. In their paper, they defined an optimization function that takes into account the leader length, the number of labeled sites, and whether groups arising from the spatial proximity of the sites or the semantics associated with them are respected. They described an ILP model and a simulated-annealing-based algorithm to compute a crossing-free labeling that maximizes the weighted sum of these three components. In contrast to our work, they allowed assigning each site to at most one group. They rewarded a labeling based on the number of consecutive ports used by labels from the same group, whereas we are, on the one hand, more strict by enforcing that grouping constraints must be respected, but, on the other hand, less strict and tolerate if the labels appear consecutively, but not necessarily at consecutive ports on the boundary. Observe that by rewarding consecutive labels if they belong to the same group, one cannot capture the (non) compliance of the labeling with overlapping groups. Gedicke et al. see combining spatial and semantic groups as an interesting direction for further research. This thesis is a step in that direction by allowing grouping constraints to overlap, thus giving the possibility to model both semantic and spatial groups in the same instance.

[1] Götzelmann et al. [GHS06b] refer to Gruber in their definition of an *ontology*, who describes it as a mechanism to conceptualize objects of a domain, for example, classes and the relationships among them [Gru93].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Finally, we want to mention the work by Klawitter et al. [KKS+23], who published, also while working on this thesis, a paper in which they propose exact and heuristic methods to visualize *geophylogenies*, i.e., phylogenetic trees where we can associate each leaf with a (geographic) site (on a map). When using the external labeling technique, they try to find a planar embedding of the phylogenetic tree on one side of the boundary, such that they can connect each site with the corresponding leaf using s-leaders with few crossings. Although they assume the phylogenetic tree to be binary, it already encodes some (implicit) grouping constraints, as closely related taxa, i.e., two taxa with a short path between their leaves, will be labeled close together on the boundary.

In conclusion, we want to underline that we see the publication of two papers closely related to the topic of this thesis while writing it as an indicator of the growing importance of grouping and ordering constraints in the literature.

1.2 Outline

This thesis consists of seven chapters. We continue in Chapter 2 by describing the fundamentals we build on in this thesis. Furthermore, we also define in this chapter the concepts and notions we will use throughout the thesis. In Chapter 3, we introduce grouping and ordering constraints and explain their intended semantics. This allows us to formally define b-Sided Constrained Boundary Labeling, the problem we are tackling in this thesis. We propose a data structure to represent our constraints and conclude this chapter with an NP-hardness result that justifies some of the assumptions we make. Chapter 4 is devoted to the main results of this thesis. We describe an algorithm to solve 1-Sided Constrained Boundary Labeling, i.e., if we are only allowed to place the labels on one side of the boundary. For 2-Sided Constrained Boundary Labeling, we provide a reduction to prove that finding a feasible labeling is NP-complete. We also analyze our algorithm and the produced labelings from a practical point of view. In Chapter 5, we describe the conducted experiments to evaluate the running time of our algorithm and the quality of the computed labelings and state the findings of the experiments. Chapter 6 presents interesting variations of 1-Sided Constrained Boundary Labeling and contains preliminary results for them. Finally, we draw our conclusions in Chapter 7, where we, on the one hand, summarize the main contributions we made with this thesis to boundary labeling but, on the other hand, also state questions that do not yet have an answer.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 2

Preliminaries

This chapter will lay the foundation for our results by introducing concepts, problems, and terminology we will use throughout this thesis. In Sections 2.1 to 2.3, we give a short introduction to set, graph, and computational complexity theory, respectively. We present the concept of external labelings, which includes boundary labelings, in Section 2.4.

2.1 Principles of Set Theory

A *set* is an unordered (finite or infinite) collection of pairwise distinct elements. Throughout this thesis, we use a calligraphic font to denote sets unless this differs from the conventions of the literature. For a set $\mathcal{A}$, we denote with $|\mathcal{A}|$ its size, i.e., the number of elements in $\mathcal{A}$. We express with $a \in \mathcal{A}$ that a is an element of $\mathcal{A}$ and with $a \notin \mathcal{A}$ that this is not the case. The set that does not contain any element is the *empty set*, indicated as $\emptyset$. A *permutation* π of $\mathcal{A}$ is an order of the elements of $\mathcal{A}$. We indicate with $\pi(a) = j$, $1 \leq j \leq |\mathcal{A}|$, that $a \in \mathcal{A}$ is located at the j-th position in the permutation π. Given two sets, $\mathcal{A}$ and $\mathcal{B}$, we will use the following operations and properties.

Intersection: $\mathcal{A} \cap \mathcal{B}$ denotes the *intersection* of $\mathcal{A}$ and $\mathcal{B}$, that is, the set containing all elements a with $a \in \mathcal{A}$ and $a \in \mathcal{B}$.

Union: $\mathcal{A} \cup \mathcal{B}$ denotes the *union* of $\mathcal{A}$ and $\mathcal{B}$, that is, the set that contains all elements a with $a \in \mathcal{A}$, or $a \in \mathcal{B}$, or both.

Difference: $\mathcal{A} \setminus \mathcal{B}$ denotes the (set) *difference* between $\mathcal{A}$ and $\mathcal{B}$, that is, the set that contains all elements a with $a \in \mathcal{A}$ but $a \notin \mathcal{B}$.

(Proper) Subset: $\mathcal{A}$ is a *subset* of $\mathcal{B}$, denoted as $\mathcal{A} \subseteq \mathcal{B}$, if and only if for every element $a \in \mathcal{A}$ we have $a \in \mathcal{B}$. If there exists a $b \in \mathcal{B}$ such that $b \notin \mathcal{A}$, then we say that $\mathcal{A}$ is a *proper subset* of $\mathcal{B}$, denoted as $\mathcal{A} \subset \mathcal{B}$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

We will use the standard notations for the sets of integers ($\mathbb{Z}$) and reals ($\mathbb{R}$). Furthermore, we denote with $\mathbb{R}_0^+$ the set of all non-negative real numbers including zero. A set consisting of other sets is called a *family* of sets. For example, the *power set* of a set $\mathcal{A}$, denoted as $2^{\mathcal{A}}$, is the family of all subsets of $\mathcal{A}$. For a family of sets $\mathscr{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, we denote with $\|\mathscr{A}\|$ the sum of the sizes of the sets in $\mathscr{A}$, i.e., $\|\mathscr{A}\| := \sum_{\mathcal{A} \in \mathscr{A}} |\mathcal{A}|$. Note that we have $|\mathscr{A}| = n$.

2.1.1 Relations on Sets

A (binary) *relation* R on a set $\mathcal{A}$ is in itself a set of ordered tuples $R \subseteq \mathcal{A} \times \mathcal{A}$. This means that a being *in relation to* b, denoted as $(a, b) \in R$, does not imply that b is also in relation to a unless $(b, a) \in R$ holds. Depending on its structure, a relation R on $\mathcal{A}$ has the following properties.

Reflexivity: The relation R is said to be *reflexive* if and only if for each $a \in \mathcal{A}$ we have $(a, a) \in R$.

Antisymmetric: We call the relation R *antisymmetric* if and only if $(a, b), (b, a) \in R$ implies that a and b are the same elements.

Transitive: The relation R is said to be *transitive* if and only if from $(a, b) \in R$ and $(b, c) \in R$, it follows that $(a, c) \in R$.

With these notions, we can define the concept of partial and total orders.

Definition 1 (Partial and Total Orders)**.** *A relation R on a set $\mathcal{A}$ is a* partial order *if R is reflexive, antisymmetric, and transitive. R is a* total order *if R is a partial order and we have, for any $a, b \in \mathcal{A}$, $(a, b) \in R$ or $(b, a) \in R$.*

We say that a total order R *extends* a partial order R' if $R' \subseteq R$ holds. To ease notation, we sometimes use *infix* notation, i.e., write aRb instead of $(a, b) \in R$.

2.2 The Fundamentals of Graph Theory

A *graph* $G = (V, E)$ consists of a (finite) set of *vertices* V and a set of *edges* $E \subseteq V \times V$. If the graph G to which the vertices and edges belong is not clear from context, we write $V(G)$ and $E(G)$. An edge $e = \{u, v\} \in E$ is said to be *undirected* if the order of its vertices is irrelevant, i.e., if $\{u, v\} = \{v, u\}$. Otherwise, we say that the edge e is *directed*, denoted as (u, v). We call a directed edge also an *arc*. If G consists only of undirected edges or directed arcs, we call it an *undirected* or *directed graph*, respectively. If G contains edges and arcs, it is a *mixed graph*. An edge $e = \{u, v\}$ *connects* the vertices u and v. If we have such an edge e, we call u and v *adjacent* (to each other) and *incident* to e. For an arc $a = (u, v)$, only u is adjacent to v, but not vice-versa unless there exists an arc $a' = (v, u)$. However, both vertices are incident to a (and a'). Furthermore, we

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

(a) (b)

Figure 2.1: A graph G before (a) and after (a) the contraction on X.

call here u the *tail* and v the *head* of a. All graphs that we consider in this thesis are, unless stated otherwise, *simple* graphs. This means, they do not contain multiple edges between the same pair of vertices and do not have self-loops, i.e., edges (or arcs) of the form $\{v, v\}$. A path $P = (v_1, \ldots, v_k)$ in G is a series of k pairwise distinct vertices such that we have $\{v_i, v_{i+1}\} \in E$ for $1 \leq i < k$. If it also holds that $\{v_k, v_1\} \in E$, then we call P a *cycle*. We call G *acyclic* if it does not contain a cycle. A graph G is *connected* if, for any two distinct vertices u and v, there is a path $P = (u, \ldots, v)$ in G. Otherwise, we say that G is *disconnected*. Let $X \subseteq V$ be a non-empty subset of vertices. If we *contract* G on X, we end up with a graph $G' = (V', E')$ with $V' := (V \setminus X) \cup \{x\}$, $x \notin V$, and $E' := \{\{u, v\} \mid u, v \in V \setminus X\} \cup \{\{x, v\} \mid v \text{ adjacent to at least one vertex of } X\}$. In other words, we first replace the vertices in X with a newly created vertex x. We connect then x with all those vertices outside X previously adjacent to at least one vertex from X. Figure 2.1 shows an example for a contraction on the green vertices.

We continue with discussing two special types of graphs, with which we will work throughout the thesis.

2.2.1 Directed Acyclic Graphs

A *directed acyclic graph* (*DAG*) $G = (V, A)$ is a directed graph that does not contain cycles. Directed graphs can be used to represent relations. Observe that for a relation R on a set $\mathcal{A}$, the directed graph $G(R) := (\mathcal{A}, R)$ encodes all the required information. For reflexive relations, we drop in this thesis the self-loops in $G(R)$. Observe that $G(R)$ is a DAG if R describes a partial order. Furthermore, any DAG encodes *some* partial order R' after possibly filling in tuples to ensure transitivity of R'.

A topological sorting of a DAG G can be interpreted as an arrangement of the vertices of G from left to right such that no arc of G goes from right to left.

Definition 2 (Topological Sorting)**.** *A* topological sorting *of a DAG G is a permutation π on its vertices such that we have $\pi(u) < \pi(v)$ for each $(u, v) \in A(G)$.*

We want to note that there can be exponentially many different topological sortings of G. Furthermore, observe that any topological sorting represents a total order that extends the partial order defined by G.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

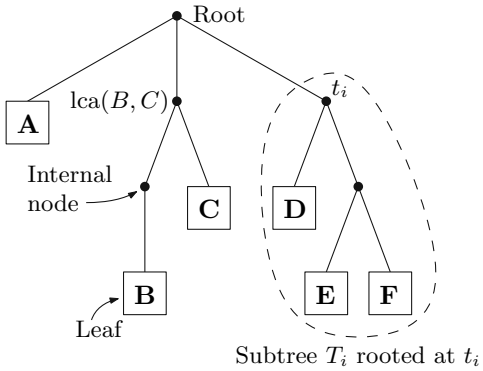


Figure 2.2: Sample of a tree T with frontier (A, B, C, D, E, F).

2.2.2 Trees

A *tree* T is an undirected, connected, and acyclic graph. Usually, one calls the vertices of T *nodes*. If we can identify a designated node as the root of the tree, we say that T is a *rooted* tree, and we usually depict T such that it "grows" downwards from its root, as in Figure 2.2. All our trees are rooted trees unless stated otherwise. Let t_i be a node of T. We call the nodes immediately below t_i the *children* of t_i and the single node above t_i the *parent* of t_i. Observe that the root does not have a parent. The *ancestors* of t_i consist of the parent of t_i and its ancestors. Note that the ancestors of t_i form a path from t_i to the root of T. Furthermore, observe that the root does not have an ancestor. A node is an *internal* node if it has at least one child else, it is a *leaf*. The set of leaves of a tree T is denoted as leaves(T). If t_i is a leaf, the *subtree* rooted at t_i, denoted as T_i, consists only of t_i. For an internal node t_i, T_i consists of t_i and the subtrees rooted at its children. Following the nomenclature of Booth and Lueker [BL76], we call the permutation obtained by reading the leaves of T from left to right the *frontier* of T. The *least common ancestor* of two leaves, l and l', of T, lca(l, l'), is the (internal) node t_i of T that is an ancestor of l and l' and has no other common ancestor of l and l' in T_i. For a set of leaves $L \subseteq$ leaves(T), we define lca(L) analogously. Note that the least common ancestor is unique. Figure 2.2 visualizes the above-introduced notions.

2.3 A Glimpse at Computational Complexity Theory

While we can state and compare the asymptotic running time or space behavior of different algorithms in terms of the $\mathcal{O}$-notation, we sometimes seek a more coarse classification. The theory of *computational complexity* is concerned with determining the *complexity class* a problem at hand falls into. For this, we only consider *decision problems* that consist of a yes/no-question, usually asking whether a given instance $\mathcal{I}$ of a problem A has a desired property. If so, we call $\mathcal{I}$ a *positive instance*. Otherwise, $\mathcal{I}$ is a *negative instance*. Note that we can phrase any optimization problem as a series of decision problems. Namely, instead of asking for the lowest cost of a solution for the problem, we

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

can subsequently ask whether there exists a solution with a cost of at most c. In the following, we will give an overview of the complexity classes we will come in contact with in this thesis. We mostly follow the nomenclature and definitions of Garey and Johnson and refer to their book for a more detailed introduction [GJ79].

2.3.1 The Complexity Classes P and NP

We call an algorithm for a problem A *polynomial-time* if we can bound its running time by a polynomial p in the size of the input instance $\mathcal{I}$, i.e., the running time is at most $p(|\mathcal{I}|)$, for some computable polynomial function p. The complexity class P contains all problems A for which there exists a deterministic polynomial-time algorithm[1] that can solve *any* instance of A. Similarly, we define the complexity class NP as the problems for which there exists a non-deterministic polynomial-time algorithm to solve them. To show that a problem belongs to NP, it suffices to show that we can encode for a given instance $\mathcal{I}$ a solution to the problem, a so-called *certificate*, in space polynomial in $\mathcal{I}$, and verify in time polynomial in $\mathcal{I}$ that it is indeed a solution.

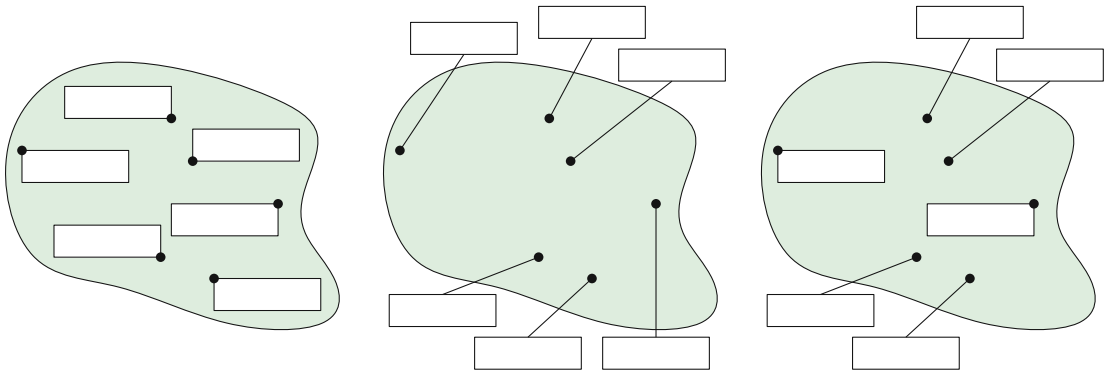
We say that we can *reduce* a problem A to another problem B if there exists a (computable) function $f : \mathrm{A} \to \mathrm{B}$ that takes an instance $\mathcal{I}_A$ of A and transforms it into an equivalent instance $\mathcal{I}_B$ of B. $\mathcal{I}_A$ and $\mathcal{I}_B$ are *equivalent* if $\mathcal{I}_B$ is a positive instance of B if and only if $\mathcal{I}_A$ is a positive instance of A. We call it a *polynomial reduction* if the transformation takes time polynomial in the size of $\mathcal{I}_A$, and the size of the instance $\mathcal{I}_B$ is polynomial in the size of $\mathcal{I}_A$. A problem A is said to be *NP-hard* if we can find for each problem B in NP a polynomial reduction from B to A. Let $\mathcal{I}$ be an instance of A. If we cannot bound the magnitude of the largest numerical value in $\mathcal{I}$ by a polynomial in the size of $\mathcal{I}$, we can, furthermore, make the following distinction. We say that A is *strongly* NP-hard if it is NP-hard even if we only consider instances $\mathcal{I}$ where we bound each numerical input value by a polynomial in the size of $\mathcal{I}$. Otherwise, we say that A is *weakly* NP-hard. Finally, if A is also in NP, we call A a (strongly/weakly) *NP-complete* problem.

2.3.2 Parameterized Complexity

Note that the definition of polynomial time looks at the instance as a whole. However, real-world instances of problems often contain some structure. For example, while road networks can be seen as a graph, each vertex will only have a limited number of incident edges, as road networks do not contain arbitrary complex road intersections. The idea behind *parameterized complexity* is to analyze the computational complexity of a problem A while taking a parameter k of the input into account, which should quantify the aforementioned structure in the input. For our road network example, the parameter could be the highest number of edges incident to a single vertex. This should allow us to give a more fine-grained classification of the computational complexity of a problem.

[1] Usually, we define the complexity classes by considering the number of steps required by a (non-)deterministic turing machine to solve the problems in this class. However, this is beyond the scope of this thesis, and we instead informally talk about the existence of an algorithm for these problems.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) Internal labeling. (b) External labeling. (c) Mixed labeling.

Figure 2.3: Comparison of different labeling styles.

One prominent class arising from parameterized complexity is *fixed-parameter-tractable* (*FPT*). We call a decision problem A *FPT with respect to a parameter* k if there exists a (deterministic) algorithm that has a running time bounded by $f(k)p(|\mathcal{I}|)$, where f is a computable function that only depends on the parameter k, and p is a computable polynomial function in the size of the instance $\mathcal{I}$. The book by Cygan et al. [CFK+15] gives an extensive introduction to parameterized algorithms in general.

2.4 External Labeling

The *labeling* process is concerned with the task of assigning textual or graphical *labels* to interesting or relevant parts of an *illustration*, which we will use in this thesis also as a synonym for image. Usually, we label feature points, so-called *sites*, of the illustration. However, labeling other objects, such as *areas* [BKPS10] or *text* [KLW14], is also possible. As the labels should provide further information about the site, it is vital that the association between a site and its label is unambiguous. Therefore, we usually aim at placing the labels close to their site without overlapping each other [HGAS05]. If we strictly obey these guidelines, we place the labels next to the sites they describe and thus inside the illustration, leading to a so-called *internal labeling*, see Figure 2.3a. Internal labelings are popular in the map labeling community [Imh75; Wol99] due to the spatial proximity between a label and its site. In the presence of many sites, large labels, or important parts of the illustration that we cannot occlude with labels, an (appealing) internal labeling might not exist. To circumvent this limitation, illustrators place the labels outside the illustration, creating a so-called *external labeling*, as shown in Figure 2.3b. As spatial proximity can no longer implicitly convey the association between a label and its site, illustrators use *leaders* to connect them explicitly. An illustration together with its labels and leaders is called a *figure* in this thesis. One can also combine internal and external labelings, as in Figure 2.3c, to create a *mixed labeling* of an illustration. In external labelings, we usually do not place the labels arbitrarily

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

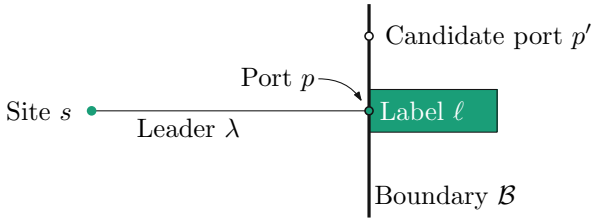


Figure 2.4: Illustration of the terminology related to boundary labeling that we use in this thesis.

outside the illustration but require that they are in a pre-defined region. This thesis deals with *boundary labeling*, where this region is a rectangular boundary around the illustration on which we must place the labels. Apart from boundary labeling, the literature also considered *contour labeling*, where we must position the labels along a (convex) shape that mimics the contour of the underlying illustration [NNR17].

2.4.1 Boundary Labeling

Boundary labeling is a restricted variant of external labeling, where we are only allowed to place the labels along a rectangular boundary. Bekos et al. [BKPS10] formally introduced this problem. Later, Bekos worked with others on a unifying taxonomy for external labeling, to which boundary labeling belongs [BNN21], and we will use their notation whenever possible. We also refer to Figure 2.4 for an illustration of the concepts introduced in this section. Let $\mathcal{B} \subset \mathbb{R}^2$ denote an axis-aligned rectangle, the so-called *boundary*, of width W and height H enclosing an illustration $\mathcal{F}$, i.e., $\mathcal{F} \subseteq \mathcal{B}$. We assume we are given n sites $\mathcal{S} = \{s_1, \ldots, s_n\}$ located inside $\mathcal{F}$ and hence also inside the bounding rectangle $\mathcal{B}$. Each site s_i, $1 \leq i \leq n$, has an axis-aligned open rectangular label ℓ_i of width w_i and height h_i associated. Furthermore, we assume that we can access the x and y coordinates of s_i using $x(s_i)$ and $y(s_i)$, respectively. Note that we have real-valued coordinates, i.e., $x(s_i), y(s_i) \in \mathbb{R}$. We use the same notion also for a port $p \in \mathcal{P}$. Unless stated otherwise, we work under the following assumptions for the sites and their labels.

Assumption 1. *The sites in $\mathcal{S}$ are in* general position*, i.e., for any two sites $s, s' \in \mathcal{S}$ with $s \neq s'$, we have $x(s) \neq x(s')$ and $y(s) \neq y(s')$.*

Assuming general position is common in the area of algorithmic geometry [BCKO08] since it allows us to describe the ideas behind an algorithm without going too much into (implementation-dependent) details. Furthermore, we can guarantee it by moving sites by some sufficiently small $\varepsilon > 0$ where necessary.

Assumption 2. *The labels have arbitrary (unbounded) width and uniform height, i.e., for all ℓ_i, $1 \leq i \leq n$, we have $h_i = h$, for some $h > 0$.*

As boundary labeling problems tend to be NP-hard if we have non-uniform height labels [BKPS10; FS16; HPL14], Assumption 2 is crucial for efficient algorithms. Further-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

more, observe that Assumption 2 is justified as the text inside the labels usually consists of a few words and should fit on a single line. Therefore, h could denote the height of a line of text [BNN21].

A *boundary labeling*, in this thesis simply called *labeling*, $\mathcal{L}$ of $\mathcal{S}$ consists of placing the labels along the boundary $\mathcal{B}$ and connecting each site $s \in \mathcal{S}$ with its label ℓ using a polyline leader λ. Depending on the number b of the sides of $\mathcal{B}$ on which we can place the labels, we call it a *b-sided* labeling. Due to the practical relevance of the problem and the nature of textual descriptions, we usually have $b \in \{1, 2\}$ and place the labels on the vertical side(s) of $\mathcal{B}$. Therefore, it is safe to ignore the width of the labels. We assume, without loss of generality, that we place the labels in a 1-sided labeling on the right side of $\mathcal{B}$. Consequently, in the 2-sided setting, we can place the labels either on the right or on the left side of $\mathcal{B}$.

The shape of the leader that connects a site with its label is usually restricted. Originally proposed by Bekos et al. [BKPS10], and later further extended [BKNS10], we can describe the style of the leader using a finite string Σ over the alphabet $\{s, p, o, d\}$. Each letter in Σ describes the layout of a segment of the leader with the following semantics [BNN21].

***s*:** A straight-line segment at an arbitrary angle.

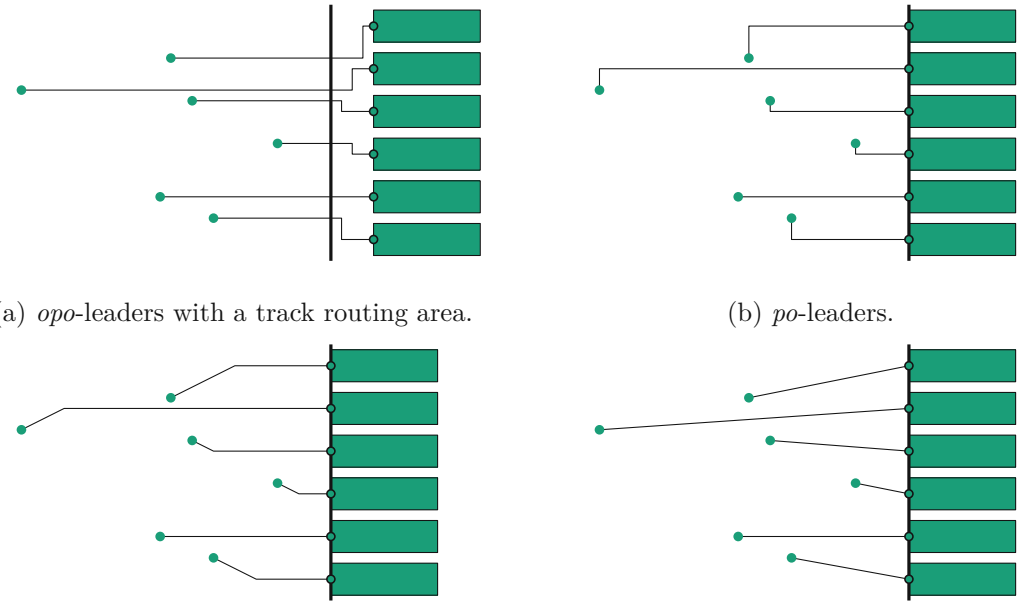
***p*:** A segment parallel to the labeling boundary. In our setting, this denotes a vertical segment.

***o*:** A segment orthogonal to the labeling boundary. In our setting, this denotes a horizontal segment.

***d*:** A segment diagonal, at some angle α, to the labeling boundary. In contrast to a straight-line segment, one usually restricts the angle α.

While Σ could be any string, *opo*-, *po*-, *do*-, and *s*-leaders are commonly used. For *opo*-leaders, we usually assume that the parallel part is outside the boundary, in a designated *track-routing area* [BKSW07]. Note that the *p*- and *d*-segments of the leaders can also be of zero length. In such a case, we say that the corresponding leader λ is *direct*. However, note that we do not refer to *s*-leaders as direct leaders. Figure 2.5 shows the different leader styles, and we refer to the user study by Barth et al. [BGNN19] for a comparison among them. In this thesis, we only deal with *po*-leaders. The place where the leader touches the label ℓ is called the *port* p of ℓ. If the port can be anywhere on the boundary of the label, we call it a *sliding-port* labeling and otherwise a *fixed-port* labeling. In this thesis, we work with fixed ports and assume that the port p of a label is placed at half the height of the label, as also in Figure 2.5. Furthermore, the literature distinguishes between *free*, *sliding*, and *fixed reference points* [BNN21]. In the first case, the label can be placed anywhere outside the illustration. This is not common in boundary labeling, as we enforce the label to be placed on the boundary. If we have sliding reference points, then the label can be placed anywhere along the boundary. Finally, for fixed reference

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) *opo*-leaders with a track routing area.

(b) *po*-leaders.

(c) *do*-leaders.

(d) *s*-leaders.

Figure 2.5: A boundary labeling of the sites from Figure 2.3 with different leader styles.

points there is only a finite set of candidate positions where we can place the label. Often there are only n candidates, and the label height h is chosen in a way such that there is no, or an equal-spaced, gap between the individual labels while guaranteeing that every text can be placed in any label [BHKN09; Kau09]. Observe that in a labeling $\mathcal{L}$, each port of a label will coincide with some reference point [BNN21]. In our thesis, we will mostly focus on a fixed and finite set of reference points. Since each label will have in this thesis a single fixed port, we use *candidate port*, or simply port if there is no risk of confusion, as a synonym for reference point. Thus, if we specify a finite set $\mathcal{P}$ of m reference points, we can think of it as specifying a set of m candidate ports, or, equivalently, defining the same m candidate positions for each label. In all these cases, a *po*-leader is completely specified by the site s and the port p. Hence, we will sometimes denote a leader λ connecting s and p as (s, p).

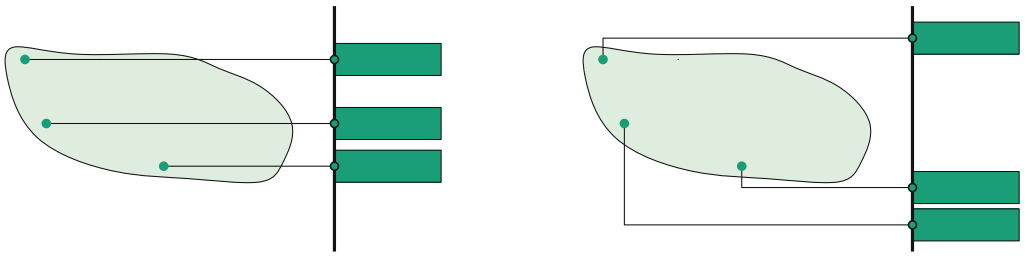
Common Constraints and Optimization Goals

The literature has proposed a variety of constraints for external labelings that also apply to boundary labelings. Those constraints often originated from interviews with illustrators. In this thesis, we use the following constraints adapted from Bekos et al. [BNN21].

C_1: No two labels overlap.

C_2: One side of each label ℓ touches the boundary $\mathcal{B}$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) (b)

Figure 2.6: An illustration, together with a labeling of its sites, that optimizes the (a) leader length and the (b) leader overlap with the illustration.

C_3: Each label ℓ faces away from the illustration. For example, if ℓ is placed on the right side of $\mathcal{B}$, the left side of ℓ touches $\mathcal{B}$.

C_4: The port p of a label ℓ is on the boundary $\mathcal{B}$ and at half the height of ℓ.

C_5: All leaders λ are of the same style (but possibly straight).

C_6: The leader λ of a site s does not cross a leader λ' or a site s' with $s \neq s'$ and $\lambda \neq \lambda'$.

While some of the constraints are already satisfied by our requirements on the input, for example, C_2 and C_3, others have to be ensured by the labeling algorithms. We call a labeling $\mathcal{L}$ *planar* if it satisfies C_1 and C_6 and *feasible* if it satisfies all of the above constraints.

Since we want to compute an appealing labeling, we are not interested in some feasible labeling $\mathcal{L}$ but in a labeling $\mathcal{L}^*$ that optimizes towards the desired quality criterion. The literature proposed several optimization goals [BNN21], where the most common ones are the following.

O_1: *Leader-Length-Minimization*, where we aim at minimizing the total leader length.

O_2: *Leader-Bend-Minimization*, where we aim at minimizing the number of bends in the leaders. This optimization criterion is not reasonable for s-leaders.

In the spirit of Benkert et al. [BHKN09], one can also define other optimization criteria. For instance, one can try to not minimize the overall leader length but the overall amount of overlap the leaders have with the underlying illustration. Observe that these two criteria are not equivalent, as Figure 2.6 shows. To support also such goals, we will in this thesis work with a general (computable) optimization function $f : \mathcal{S} \times \mathcal{P} \to \mathbb{R}_0^+$ that evaluates a site-port combination independent of the remaining labeling. As for *po*-leaders the leader $\lambda = (s, p)$ is uniquely defined by the site-port combination, we write $f(\lambda)$ to denote $f(s, p)$. Furthermore, we use slight abuse of notation and define $f(\mathcal{L}) := \sum_{\lambda \in \mathcal{L}} f(\lambda)$ for a feasible labeling $\mathcal{L}$. If $\mathcal{L}$ is not feasible, we define $f(\mathcal{L}) := \infty$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Finally, we want to point out that O_1, O_2, and the basic feasibility-question, can be modeled using the following functions.

Feasibility: Since we only want a feasible labeling, we can set $f(s,p) := 1$, for all $s \in \mathcal{S}$ and $p \in \mathcal{P}$, and observe that a feasible labeling $\mathcal{L}$ has a cost of $f(\mathcal{L}) = n$.

O_1: To obtain a leader-length-minimal labeling with *po*-leaders, we use the *Manhattan Distance* (L_1-norm) as our optimization function, i.e., $f(s,p) := |y(p) - y(s)| + |x(p) - x(s)|$. Observe that in a 1-sided setting, the position of the site dictates the horizontal distance of the leader. Thus, we can use $f(s,p) := |y(p) - y(s)|$.

O_2: To obtain a bend-minimal labeling, we have to penalize every bend, i.e.,

$$f(s,p) := \begin{cases} 0 & \text{if } y(s) = y(p), \\ 1 & \text{otherwise.} \end{cases}$$

Using these ingredients, we can formally define the b-Sided Boundary Labeling problem for $b \in \{1,2\}$.

Problem 1 (b-Sided Boundary Labeling – based on [BNN21])**.**

Given: *A set of n sites $\mathcal{S} = \{s_1, \ldots, s_n\}$, a set of m candidate ports $\mathcal{P} = \{p_1, \ldots, p_m\}$ on b sides of the boundary $\mathcal{B}$ (around $\mathcal{S}$), a label height $h > 0$, and a computable function $f : \mathcal{S} \times \mathcal{P} \to \mathbb{R}_0^+$.*

Task: *Find a feasible b-sided po-labeling $\mathcal{L}$ on the candidate ports $\mathcal{P}$ with labels of height h that minimizes $f(\mathcal{L})$.*

Since we provide a set of candidate ports $\mathcal{P}$, b-Sided Boundary Labeling uses fixed reference points. However, one can easily define b-Sided Boundary Labeling with sliding reference points. Throughout this thesis, we implicitly assume $n \leq m$, as otherwise an instance is doomed to be infeasible.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Grouping and Ordering Constraints and Their Representations

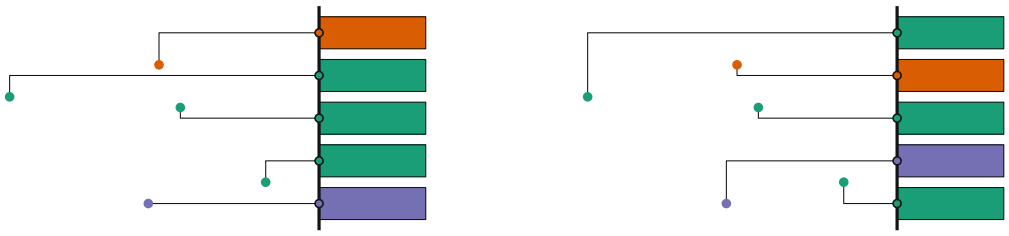
In Chapter 2, we discussed the b-Sided Boundary Labeling problem, a well-studied problem for which polynomial-time algorithms exist. The overall aim of this thesis is to incorporate *semantic constraints* into boundary labeling that arise from the meaning associated with the sites, in particular, grouping and ordering constraints. Since we can interpret a labeling in the one-sided setting as a permutation π of the sites, reading the labels from top to bottom, we will define grouping and ordering constraints by enforcing certain patterns in π. However, we will also describe ways to extend these constraints to the two-sided setting.

We start with Sections 3.1 and 3.2, where we define grouping and ordering constraints, respectively. While stating such constraints should be intuitive from the perspective of an illustrator, their lack of structure makes it hard to use them efficiently in algorithms. Therefore, we also look into data structures that allow us to represent such constraints more effectively and present them in Section 3.3. They allow us to describe in Section 3.4 a polynomial-time preprocessing procedure to check whether the given constraints are respectable by the sites. We conclude this chapter with Section 3.5, where we formally define the problem we investigate in this thesis and discuss a preliminary NP-hardness result.

3.1 Grouping Constraints

A *grouping constraint* $\emptyset \subset \mathcal{G} \subseteq \mathcal{S}$ enforces that the labels for the sites in $\mathcal{G}$ must appear consecutively on the boundary. This means that the labels for the group $\mathcal{G}$ cannot be

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) (b)

Figure 3.1: Two different labelings of a set of sites with grouping constraints: Labeling (a) respects the constraints whereas Labeling (b) does not. Each color represents a grouping constraint.

separated by a label for a site $s \notin \mathcal{G}$. Figure 3.1 visualizes this. However, observe that the order of the labels within a group is a priori unrestricted. Only in the presence of multiple non-disjoint grouping constraints, the order of the labels inside the group can be partially restricted. Consider for example the set of sites $\mathcal{S} = \{s_1, s_2, s_3\}$ together with the grouping constraints $\mathscr{G} = \{\{s_1, s_2\}, \{s_2, s_3\}\}$. As the two groups overlap, the label ℓ_2 (for s_2) must appear between ℓ_1 and ℓ_3 (for s_1 and s_3, respectively) in any feasible labeling $\mathcal{L}$ that respects the grouping constraints.

For multi-sided boundary labeling problems, we enforce that we label a group of sites entirely on one side of the boundary. The reason behind this is that labels on different sides of the boundary usually have a (comparably) large white space between them. Especially for two opposite sides, this makes the group less clear and no longer allows an illustrator to enrich a group of sites with further information next to the labels.

Assumption 3. *In a multi-sided boundary labeling problem all sites for a group must be labeled on the same side of the boundary.*

In the remainder of this section, we assume, unless stated otherwise, a one-sided boundary labeling problem. However, the results can easily be extended to multi-sided settings.

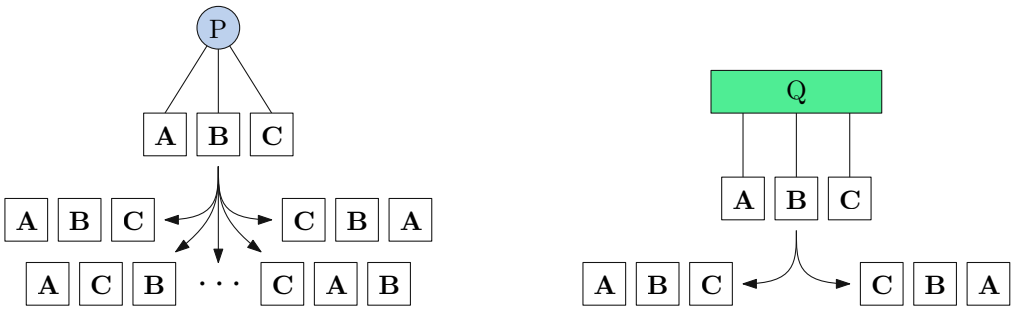
We say that a labeling $\mathcal{L}$ *respects* the grouping constraints $\mathscr{G}$ if the labels for any group $\mathcal{G} \in \mathscr{G}$ are consecutive in $\mathcal{L}$.[1] The grouping constraints $\mathscr{G}$ are *respectable* for a set of sites $\mathcal{S}$ if there exists a labeling that respects them. Observe that not all grouping constraints are respectable, as the following example shows.

$$\mathcal{S} = \{s_1, s_2, s_3\} \qquad \mathscr{G} = \{\{s_1, s_2\}, \{s_2, s_3\}, \{s_1, s_3\}\}$$

We can see that in any of the six possible permutations π of $\mathcal{S}$, the grouping constraint that involves the first and the last site, according to π, is not respected. Thus, certainly, there does not exist a labeling of $\mathcal{S}$ that respects all grouping constraints.

[1] Note that this does *not* enforce the labels to occupy *consecutive* ports. There may be ports between two labels of the same group, but they must not be used by labels outside the group.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) A P-node and its allowed operations.

(b) A Q-node and its allowed operations.

Figure 3.2: The two different types of nodes in a PQ-Tree.

For our problem at hand, we assume as input a family $\mathscr{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ of k grouping constraints. However, having a family of sets does not reflect the implications of intersecting grouping constraints on the set of feasible labelings. Since we can interpret a boundary labeling $\mathcal{L}$ as a permutation of the sites $\mathcal{S}$ and a grouping constraint as a way of restricting the set of potential permutations, it becomes natural to consider a data structure that represents constrained permutations of a set.

3.1.1 PQ-Trees

Booth and Lueker defined PQ-Trees to help solve the problem of finding so-called *permissible permutations* [BL76] of a set $\mathcal{U}$. They define this as all those permutations π of $\mathcal{U}$, where, for a family of sets $\mathscr{W}$, the elements of each $\mathcal{W} \in \mathscr{W}$ appear consecutively in π. Observe the close relation to our problem. Effectively, we have $\mathcal{U} = \mathcal{S}$ and $\mathscr{W} = \mathscr{G}$. For a given set $\mathcal{U}$, a *PQ-Tree* is a rooted tree T with a bijection between the leaves of T and the elements of $\mathcal{U}$, i.e., each leaf of T represents exactly one element of $\mathcal{U}$ and each element of $\mathcal{U}$ is represented by exactly one leaf of T. Each internal node t of T is either a P-node or a Q-node, which places different constraints on the order of its children. If t is a *P-node*, we are free to permute the children of t in any order. On the other hand, if t is a *Q-node*, we can only reverse the order of its children. Figure 3.2 visualizes the difference between those two types of nodes. Similar to Booth and Lueker, we visualize P-nodes as circles and Q-nodes as rectangles. Throughout this thesis, we assume, unless otherwise stated, that each P-node has at least two children and each Q-node has at least three children so that the operations have an effect. Booth and Lueker called such PQ-Trees *proper* [BL76]. A consequence of this assumption is that $|T| = \mathcal{O}(|\mathcal{U}|)$ holds [JLCZ20]. For a PQ-Tree T, we will denote with t_r its root.

3.1.2 On the Representability of Grouping Constraints via PQ-Trees

Having now discussed PQ-Trees as a way of representing our grouping constraints $\mathscr{G}$, we should verify that we can represent at least all interesting, i.e., respectable, grouping constraints with them. To do that, we first present the consecutive ones property of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

binary matrices. A binary matrix M has the *consecutive ones property* if there exists a permutation of the rows such that the ones in each column are consecutive [FG65]. We will now define a matrix that captures the grouping constraints $\mathscr{G}$ over a set of sites $\mathcal{S}$. Without loss of generality, we assume that $\mathscr{G} \neq \emptyset$ holds. This can be satisfied by adding the trivial grouping constraint $\mathcal{S}$ to $\mathscr{G}$. Let $M(\mathcal{S}, \mathscr{G})$ be a binary matrix of dimensionality $n \times k$, with $n = |\mathcal{S}|$ and $k = |\mathscr{G}|$. An entry $m_{i,j} \in M(\mathcal{S}, \mathscr{G})$ is one if the site $s_i \in \mathcal{S}$, $1 \leq i \leq n$, appears in the group $\mathcal{G}_j \in \mathscr{G}$, $1 \leq j \leq k$, i.e., if $s_i \in \mathcal{G}_j$ holds, and otherwise $m_{i,j}$ is zero. We call the matrix $M(\mathcal{S}, \mathscr{G})$ the *sites vs. groups* matrix (for $\mathcal{S}$ and $\mathscr{G}$). Intuitively, we can see the following equivalence. Each ordering π of the rows of $M(\mathcal{S}, \mathscr{G})$ represents a (set of) possible labeling(s), not necessarily feasible, in which the order of the labels is given by the order of the rows of $M(\mathcal{S}, \mathscr{G})$ under π. If the ones in column j, $1 \leq j \leq k$, of $M(\mathcal{S}, \mathscr{G})$ are consecutive under π, then the grouping constraint $\mathcal{G}_j$ is satisfied by the labeling induced by π. We prove this intuition formally in Lemma 3.1.

Lemma 3.1. *Let $\mathscr{G}$ be a non-empty family of grouping constraints for the sites $\mathcal{S}$. $\mathscr{G}$ is respectable for $\mathcal{S}$ if and only if the sites vs. groups matrix $M(\mathcal{S}, \mathscr{G})$ has the consecutive ones property.*

Proof. We show both directions separately.

($\Rightarrow$) We assume that the grouping constraints $\mathscr{G}$ are respectable. This means there exists a (not necessarily feasible) labeling $\mathcal{L}$ that respects all grouping constraints. We order the rows of $M(\mathcal{S}, \mathscr{G})$ according to the order imposed by $\mathcal{L}$ on the sites $\mathcal{S}$ and obtain thus a permutation π. As $\mathcal{L}$ is a witness labeling for the respectability of $\mathscr{G}$, each group $\mathcal{G}_j$, $1 \leq j \leq k$, is not intersected by a label for a site that is not part of $\mathcal{G}_j$. Therefore, for each group $\mathcal{G}_j$, the ones in the jth column of $M(\mathcal{S}, \mathscr{G})$ must be consecutive. Consequently, π witnesses that $M(\mathcal{S}, \mathscr{G})$ has the consecutive ones property.

($\Leftarrow$) We assume that $M(\mathcal{S}, \mathscr{G})$ has the consecutive ones property. This means there exists a permutation π of the rows, i.e., the sites, such that the ones in each column, i.e., for each group, are consecutive. If we order the labels according to the order of their respective sites in π and create the corresponding (not necessarily feasible) labeling $\mathcal{L}$, $\mathcal{L}$ will respect all the grouping constraints. If not, this would mean that we have a group $\mathcal{G}_j$, $1 \leq j \leq k$, that is intersected by a label for a site $s_i \notin \mathcal{G}_j$ outside the group. However, by our definition of $M(\mathcal{S}, \mathscr{G})$, $m_{i,j} = 0$ must hold, and since the group $\mathcal{G}_j$ is intersected by s_i, we know that $m_{i_1,j} = m_{i_2,j} = 1$ holds, for some $\pi(i_1) < \pi(i) < \pi(i_2)$, $1 \leq i, i_1, i_2 \leq n$. This would mean that π is not a witness ordering for $M(\mathcal{S}, \mathscr{G})$ having the consecutive ones property – Contradiction. Therefore, $\mathcal{L}$ must be a witness for the respectability of $\mathscr{G}$. □

Booth and Lueker [BL76] propose an algorithm to check whether a binary matrix M has the consecutive ones property. Their algorithm internally uses a PQ-Tree to keep track of the allowed row permutations and can be easily adapted to output this PQ-Tree in the end, if it exists. Hence, we get the following corollary.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

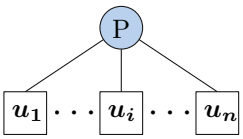


Figure 3.3: The universal PQ-Tree for the set $\mathcal{U} = \{u_1, \ldots, u_n\}$. Figure adapted from Booth and Lueker [BL76, Figure 5].

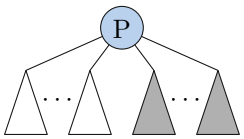
Corollary 3.1 (Together with [BL76, Theorem 6])**.** *For each family of respectable grouping constraints, there exists a PQ-Tree that represents them.*

3.1.3 Creating PQ-Trees

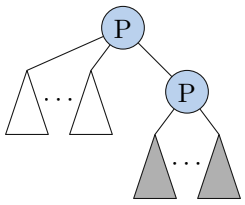
The literature has widely applied PQ-Trees, for example, to recognize and extend (partial) representations of interval graphs [KKO+17], detect permutations of patterns in genomes [LPW05], solve the Traveling Salesperson Problem where the set of tours is constrained via a PQ-Tree [BDW98], and now also in the context of grouping constraints in boundary labeling. Furthermore, the concept of PQ-Trees has been extended to include further types of nodes, such as nodes that fix the order of their children, yielding so-called *FPQ-Trees* [LRT21], or failure nodes to indicate parts in the PQ-Tree where the constraints cannot be ensured, so-called *PQR-Trees* [MPT98]. Despite the widespread use of PQ-Trees, Fink et al. [FPR21] have identified a lack of (correct) implementations of PQ-Trees. One reason for this could be that while understanding PQ-Trees, their purpose, and their semantics is easy, efficiently creating them is far from trivial. Therefore, we want to sketch in this section the crux of the algorithm proposed by Booth and Lueker [BL76] to create PQ-Trees, as we think it facilitates understanding the upcoming chapters where we use and manipulate PQ-Trees.

In the algorithm, we start with the so-called *universal tree* T, i.e., the trivial PQ-Tree that allows all $n!$ possible permutations. It is visualized in Figure 3.3 and consists of a single P-node having all n elements of $\mathcal{U}$ as its children. We then iteratively ensure that in all permutations π allowed by T, the elements for some new $\mathcal{W} \in \mathscr{W}$ appear consecutively in π. To do that, we have to go for each $\mathcal{W} \in \mathscr{W}$ through the following two phases. For the first phase, the so-called *bubble-up* phase, let t' be a node of T and T' the subtree of T rooted at t'. We mark t' as *full*, *empty*, or *partial*, depending on whether the leaves of T' contain only elements of $\mathcal{W}$, no element of $\mathcal{W}$, or some elements of $\mathcal{W}$ and some not in $\mathcal{W}$, respectively. We do this in a bottom-up procedure for all nodes t' in the so-called *pruned pertinent subtree* $T_\mathcal{W}$ of T. $T_\mathcal{W}$ is the subtree rooted at the $\text{lca}(\mathcal{W})$ that does not contain leaves that represent sites not in $\mathcal{W}$ [BL76]. On the way up, we mark each node accordingly and store there further information, including its children, the left- and rightmost child, and its type. Having computed this information for all nodes in $T_\mathcal{W}$, we perform the actual manipulation of the PQ-Tree in the second phase, the *reduce* phase. To do that, we go a second time bottom-up through the pruned pertinent subtree and use the before-computed information to check for each node t' in $T_\mathcal{W}$ whether it matches one of several patterns. We identify with these patterns crucial

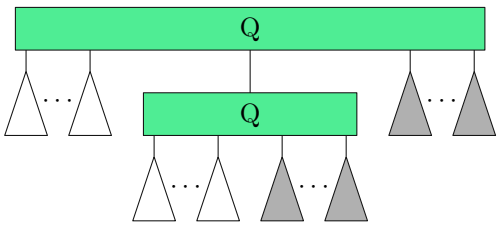
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



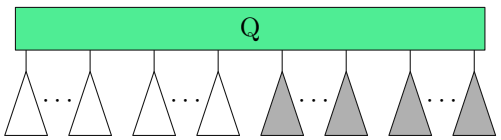
(a) A pattern for a P-node.



(b) Replacement for the pattern of Figure 3.4a.



(c) A pattern for a Q-node.



(d) Replacement for the pattern of Figure 3.4c.

Figure 3.4: Patterns for a (a) P-node and a (c) Q-node with their corresponding replacements. Empty nodes are white, and full nodes are gray. Figures adapted from Booth and Lueker [BL76, P-node: Figure 8, Q-node: Figure 14].

parts in the (pruned pertinent) subtree that need to be adjusted to ensure that in all permutations of $\mathcal{U}$ allowed by T, the elements of $\mathcal{W}$ appear consecutively. Each pattern is accompanied by a corresponding replacement t'': If the pattern matches, we replace t' with t''. Figure 3.4 contains two such patterns. In the pattern from Figure 3.4a, we, for example, check if t' is the root of $T_{\mathcal{W}}$, a P-node, and whether some children of t' are full, i.e., all their leaves are from $\mathcal{W}$. If so, we must make sure that in any permutation of its children, the full children are next to each other by introducing a further P-node as in Figure 3.4b. On the other hand, if t' is, for example, a Q-node with some full children, and one of them is another Q-node that also contains some full children, as in Figure 3.4c, we must ensure that both Q-nodes are simultaneously inversed as otherwise the elements of $\mathcal{W}$ are not necessarily consecutively. We can ensure this by merging the two Q-nodes of Figure 3.4c into a single Q-node, as in Figure 3.4d.

However, if, given the information, t' contrary to the expectations does not match a pattern, then the set of permissible permutations is empty, and we return a designated tree that does not allow any permutation of $\mathcal{U}$ at all. After applying those steps for all sets $\mathcal{W} \in \mathscr{W}$, we end up with a PQ-Tree T that represents all permissible permutations of $\mathcal{U}$ given $\mathscr{W}$.

While we have only scratched the surface of the algorithm, it should become clear that it is far from trivial to implement it. Not only are there almost a dozen different patterns, some of which also have variations, but without the information obtained in the first phase, we could not match (and replace) patterns efficiently. Thanks to pre-computing

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

relevant information in the bubble-up phase, we can obtain the PQ-Tree T that represents all permissible permutations of a set $\mathcal{U}$ with respect to $\mathscr{W}$ in $\mathcal{O}(n + |\mathscr{W}| + \|\mathscr{W}\|)$ time, that is, linear in the input size, plus the sum of the sizes of the sets in $\mathscr{W}$ [BL76].

3.2 Ordering Constraints

While grouping constraints can only implicitly enforce an order on the labels of the sites, ordering constraints are tailored for that. An *ordering* constraint (s, s'), for $s, s' \in \mathcal{S}$, enforces that we have, for the leaders $\lambda = (s, p)$ and $\lambda' = (s', p')$, $y(p') \leq y(p)$, i.e., the label for s' must not appear above the label for s. For the ordering constraints, we expect as input a relation $\prec \subseteq \mathcal{S} \times \mathcal{S}$ containing r tuples, that represent a partial order on the sites. Observe that we can, therefore, also understand the ordering constraints as satisfied, if the total order on the labels, given by reading them from top to bottom, extends the partial order on the sites given by $\prec$. Throughout the thesis, we implicitly assume that $\prec$ contains the required tuples for being a reflexive and transitive relation. Furthermore, we use infix notation, i.e., $s \prec s'$. Recall that we can represent any relation as a directed graph. Therefore, we can trivially encode $\prec$ in a directed (acyclic) graph $G(\prec)$. We write G instead of $G(\prec)$ if there is no risk of confusion.

Analog to grouping constraints, we say that a labeling $\mathcal{L}$ *respects* $\prec$ if all ordering constraints in $\prec$ are satisfied. Furthermore, we say that $\prec$ is *respectable* for the sites $\mathcal{S}$ if there exists a labeling $\mathcal{L}$ of $\mathcal{S}$ that respects $\prec$. Note that we do not require that $\mathcal{L}$ is feasible. For example, $\mathcal{L}$ might contain crossing leaders. Observe that this is equivalent to finding a total order on the sites that extends the partial order $\prec$.

Finally, there are several natural ways to define an ordering constraint $s \prec s'$ in the case of a multi-sided boundary labeling problem. In the spirit of Niedermann et al. [NNR17], one could, instead of the y-coordinates of the used ports, consider the clockwise ordering of the labels, starting at the upper-right corner of the boundary. However, the rather rigid position of the labels on a rectangular boundary does not allow for a sensible interpretation of the labels in their clockwise order. Furthermore, we read in the Western world from top to bottom and, therefore, would assume that the notion of "before" is also from top to bottom. As another possible extension, one could, in the presence of $s \prec s'$, also enforce $y(p') \leq y(p)$ for multiple sides. However, this interpretation is not very suited due to the spatial distance labels on different sides of the boundary usually have. Therefore, it is not so critical to ensure that the port p used by s is further up than the port p' used by s' if they are on different sides. In the end, it is unlikely that they are looked at right after each other since we might prefer to process each side of the boundary individually. Thus, it is more important, that we fulfill the ordering constraints for those labels that are near each other and hence are processed contemporaneously, i.e., the labels on the same side of the boundary. Therefore, we assume in a multi-sided boundary labeling problem the following.

Assumption 4. *In a multi-sided boundary labeling problem, the ordering constraints must be fulfilled for each side independently, i.e., for $s \prec s'$, if s and s' are labeled at p*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

and p', respectively, and p and p' are on the same side of the boundary, then we must have $y(p') \leq y(p)$. However, if they are labeled on different sides of the boundary, this constraint has no effect.

Observe that the interpretation given in Assumption 4 is supported by the motivating example of Figure 1.1b that labels the rings of the tree.

3.3 The PQ-A-Graph

Now that we have convinced ourselves that we can represent any set of respectable grouping and ordering constraints by PQ-Trees and directed graphs, respectively, we want to combine them to represent our constraints by a single data structure. Let $\mathcal{S}$ be a set of n sites, $\mathscr{G}$ be a family of k grouping constraints, and $\prec$ be a relation consisting of r ordering constraints. We assume, without loss of generality, that the constraints are respectable. If the constraints are not respectable there is no need to represent them, as any instance will be infeasible. We know that we can represent the grouping constraints $\mathscr{G}$ using a PQ-Tree T. Furthermore, we can represent the ordering constraints $\prec$ by a directed graph G. We now combine these two data structures into the *PQ-A-Graph*.

Definition 3 (PQ-A-Graph)**.** *Let $\mathcal{S}$ be a set of sites, $\mathscr{G}$ be a family of respectable grouping constraints, and $\prec$ be a relation of ordering constraints. The* PQ-A-Graph $\mathcal{T} = (T, A)$ *consists of the PQ-Tree T that represents the grouping constraints $\mathscr{G}$, on whose leaves we embed the directed graph $G(\prec)$ with arcs A, which represents the ordering constraints $\prec$.*

Observe that $\mathcal{T}$ is a mixed graph. Note that, unless stated otherwise, we implicitly refer with a *subtree* of $\mathcal{T}$ to a subtree in the underlying PQ-Tree T. Analogously, a *subgraph* of $\mathcal{T}$ is a subgraph of the underlying directed graph $G(\prec)$. Furthermore, when we refer to the site s in $\mathcal{T}$, we implicitly refer to the leaf l of $\mathcal{T}$ that represents s. Consequently, we will address (the arc representing) an ordering constraint $s \prec s'$ using (l, l') or (s, s') interchangeably.

We want to point out that the PQ-A-Graph does not preserve the structure of the grouping constraints, i.e., not every group $\mathcal{G} \in \mathscr{G}$ appears as a subtree of $\mathcal{T}$. Consider the sites $\mathcal{S} = \{A, B, C, D, E\}$ together with the grouping constraints $\mathscr{G} = \{\{B, C, D\}, \{D, E\}\}$. The corresponding PQ-A-Graph $\mathcal{T}$ is visualized in Figure 3.5. Note that $\mathcal{T}$ effectively consists only of the PQ-Tree T due to the lack of ordering constraints. Although a grouping constraint might not be reflected as a subtree, we can interpret each subtree T' of $\mathcal{T}$ as a grouping constraint, since in any labeling that adheres to the constraints represented by $\mathcal{T}$, the sites represented by leaves(T') will appear consecutively on the boundary. We call these the *canonical groups* of $\mathcal{T}$. The canonical groups of the PQ-A-Graph from Figure 3.5 are $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{B, C\}$, $\{B, C, D, E\}$, and $\mathcal{S}$. Observe that in any PQ-A-Graph $\mathcal{T}$, $\mathcal{T}$ and any site $s \in \mathcal{S}$ form a canonical group. In the following, we want to argue the properties of PQ-A-Graphs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

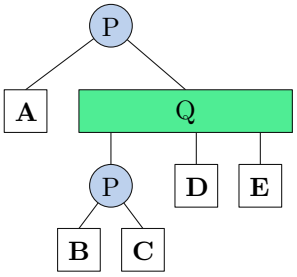


Figure 3.5: The PQ-A-Graph $\mathcal{T}$ for the grouping constraints $\{\{B, C, D\}, \{D, E\}\}$.

Lemma 3.2. *Let $\mathcal{S}$ be a set of sites, $\mathscr{G}$ be a family of grouping constraints, and $\prec$ be a relation of ordering constraints. Assume that the constraints are respectable for $\mathcal{S}$. We can create the corresponding PQ-A-Graph $\mathcal{T}$ in $\mathcal{O}(n + k + r + \|\mathscr{G}\|)$ time, where n is the number of sites, k the number of grouping constraints, and r the number of ordering constraints.*

Proof. We start by reminding the reader that in order to obtain Corollary 3.1, we observed that Booth and Lueker propose an algorithm to check whether a binary matrix M has the consecutive ones property. This algorithm can be modified to return the corresponding PQ-Tree on success. By Lemma 3.1, we know that the sites vs. groups matrix $M(\mathcal{S}, \mathscr{G})$ has the consecutive ones property, as $\mathscr{G}$ is respectable for $\mathcal{S}$ by assumption, i.e., we are guaranteed to obtain a PQ-Tree. Since their algorithm decomposes M into its columns to build the PQ-Tree out of those, which for $M(\mathcal{S}, \mathscr{G})$ would correspond to the sets $\mathcal{G} \in \mathscr{G}$, we do not need to compute $M(\mathcal{S}, \mathscr{G})$ but can directly work with $\mathscr{G}$. Hence, we can obtain the PQ-Tree used to build $\mathcal{T}$ in time $\mathcal{O}(n + k + \|\mathscr{G}\|)$, i.e., the running time of the algorithm proposed by Booth and Lueker [BL76, Theorem 6].

In the following, we assume that we maintain a map that returns for each site $s \in \mathcal{S}$ the corresponding leaf in T. Since we never add or remove a leaf, maintaining this list does not increase the asymptotic running time of creating T.

To finish the creation of $\mathcal{T}$, we have to enrich T by the ordering constraints $\prec$. Let $s \prec s'$ be one of those constraints. Since we can find the leaves for s and s' in T in constant time using our lookup table, adding the arc (s, s') to T takes $\mathcal{O}(1)$ time. This sums up to $\mathcal{O}(r)$ and together with above arguments we get $\mathcal{O}(n + k + r + \|\mathscr{G}\|)$. □

Lemma 3.3. *Let $\mathcal{S}$ be a set of sites, $\mathscr{G}$ be a family of grouping constraints, and $\prec$ be a relation of ordering constraints. Assume that the constraints are respectable for $\mathcal{S}$ and let $\mathcal{T}$ be the corresponding PQ-A-Graph. $\mathcal{T}$ uses $\mathcal{O}(n + r)$ space, where n is the number of sites and r is the number of ordering constraints in $\prec$.*

Proof. To transform the PQ-Tree T into the PQ-A-Graph $\mathcal{T}$, we need to embed the (arcs of the) directed graph $G(\prec)$ into T. We can do this using adjacency sets, i.e., adjacency lists where we use sets to store the neighbors of the nodes. This gives us

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

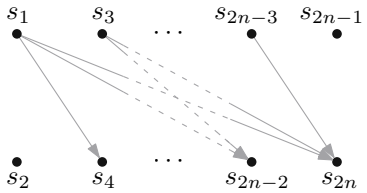


Figure 3.6: A set of sites together with some ordering constraints, visualized in gray, that must be stored on the leaves of the corresponding PQ-A-Graph.

constant time look-up while needing $\mathcal{O}(n+r)$ space. So what remains to analyze is the space consumption of T. Recall that we assume that T (and hence also $\mathcal{T}$) is proper, which means that any P-node has at least two and any Q-node at least three children, respectively, i.e., there are no (chains of) nodes with a single child. Hence, $\mathcal{T}$ uses $\mathcal{O}(n)$ space [JLCZ20]. The overall space consumption of $\mathcal{O}(n+r)$ follows by combining the individual ones. □

One could argue that embedding the directed graph $G(\prec)$ (solely) on the leaves of T is a limitation of the PQ-A-Graph, as this results in storing potentially redundant ordering constraints. That is, if several arcs in $\mathcal{T}$ go from one part of the underlying PQ-Tree T into another one, they are all present, even though we could only store a single arc further up in T. While this is true in most cases, we can also find instances where this is not possible. Consider the following example. Let $\mathcal{S}$ be a set of $2n$ sites together with the ordering constraints $\{(s_{2i-1}, s_{2j}) \mid 1 \leq i \leq n,\ i < j \leq n\}$. The sites can be arranged in a ladder-like layout, as seen in Figure 3.6. The resulting PQ-A-Graph $\mathcal{T}$ has a single P-node at its root and all the leaves as its immediate children, like the universal PQ-Tree from Figure 3.3. So, there is no way to embed arcs further up in the tree. In addition, observe that there is no pre-defined order among the sites from the upper side of the ladder as well as among the ones from the lower side, respectively. Furthermore, since also the sites from different sides of the ladder can be interleaved relatively freely, we cannot infer implicit grouping constraints so that $\mathcal{T}$ would consist of more internal nodes. Hence, we arrive at the following observation.

Observation 3.1. *The $\mathcal{O}(n+r)$ space consumption of a PQ-A-Graph, where n is the number of sites, and r is the number of ordering constraints, is worst-case optimal.*

3.4 An Efficient Preprocessing Procedure

Throughout this chapter we often assumed that the given constraints are respectable for a set of sites $\mathcal{S}$. This assumption made our lives easier, as, for example, the PQ-A-Graph is only defined for respectable constraints. In this section, we want to justify this assumption by proving that we can efficiently check whether some given constraints $\mathcal{C} = (\mathscr{G}, \prec)$ are respectable for $\mathcal{S}$. To do that, we will combine different results from the literature. More concretely, we use Lemma 3.1 and the underlying algorithm by Booth and Lueker [BL76],

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

which we already used to prove Lemma 3.2. We will combine this result by showing that the respectability of $\mathcal{C}$ by $\mathcal{S}$ is equivalent to solving the Reorder problem on T and $G(\prec)$, i.e., $\mathcal{T}$.

Problem 2 (Reorder [KKO+17])**.**

Given: *A PQ-Tree T and a partial order $\prec$.*

Question: *Can we re-order the leaves of T such that the resulting total order induced by the frontier of T extends the partial order $\prec$?*

Observe that this is equivalent to the constraints expressed by the PQ-A-Graph $\mathcal{T}$ being respectable for the sites $\mathcal{S}$. This follows readily from the definition of respectability, as we can re-order in a witness labeling the leaves of $\mathcal{T}$ such that all embedded arcs go from left to right, i.e., we found a topological sorting on the leaves that extends the partial order into a total order. We use this equivalence now to prove Lemma 3.4.

Lemma 3.4. *Let $\mathcal{S}$ be a set of sites, $\mathscr{G}$ be a family of grouping constraints, and $\prec$ be a relation of ordering constraints. We can check whether the constraints $\mathcal{C} = (\mathscr{G}, \prec)$ are respectable for $\mathcal{S}$ in $\mathcal{O}(n + k + r + \|\mathscr{G}\|)$ time, where n is the number of sites, k is the number of grouping, and r is the number of ordering constraints.*

Proof. Recall that Booth and Lueker [BL76] propose an algorithm to check whether a binary matrix M has the consecutive ones property. Using this algorithm, we can check in $\mathcal{O}(n + k + \|\mathscr{G}\|)$ time whether the sites vs. groups matrix $M(\mathcal{S}, \mathscr{G})$ has the consecutive ones property. We showed in Lemma 3.1 that this is a sufficient and necessary property for respectability of the grouping constraints $\mathscr{G}$ by the sites $\mathcal{S}$. If this algorithm outputs that $M(\mathcal{S}, \mathscr{G})$ does not have the consecutive ones property, we are done. So assume that $M(\mathcal{S}, \mathscr{G})$ has the consecutive ones property and let T be the PQ-Tree representing the grouping constraints. Recall that we can modify the algorithm by Booth and Lueker to return it if it exists without spending additional time.

What remains to check is whether T allows for a permutation that extends the partial order $\prec$. As argued above, this is equivalent to the instance $(T, \prec)$ of the Reorder problem. Klávic et al. show that we can solve this problem in $\mathcal{O}(n + r)$ time [KKO+17, Proposition 2.4]. The claimed running time follows then readily. □

Observe that the algorithm implicitly defined by merging the two steps mentioned in the proof of Lemma 3.4 can serve as a preprocessing routine for all upcoming algorithms. If it turns out that the constraints $\mathcal{C}$ are not respectable for $\mathcal{S}$, we know that no feasible labeling will exist either. Since this preprocessing routing has the same time complexity as creating the PQ-A-Graph $\mathcal{T}$ out of $\mathcal{C}$, we will not consider it further in the upcoming proofs. Therefore, in the remainder of this thesis, we implicitly work under Assumption 5 unless stated otherwise.

Assumption 5. *The constraints $\mathcal{C} = (\mathscr{G}, \prec)$, with grouping constraints $\mathscr{G}$ and ordering constraints $\prec$, are respectable for a set of sites $\mathcal{S}$.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.5 The b-Sided Constrained Boundary Labeling Problem

Having now defined grouping and ordering constraints, and a data structure to represent them that also allows us to quickly identify trivial infeasible instances, we are ready to state the main problem we will tackle in this thesis, the b-Sided Constrained Boundary Labeling problem.

Problem 3 (b-Sided Constrained Boundary Labeling)**.**

Given: *A set of n sites $\mathcal{S} = \{s_1, \ldots, s_n\}$, a set of m candidate ports $\mathcal{P} = \{p_1, \ldots, p_m\}$ on b sides of the boundary $\mathcal{B}$ (around $\mathcal{S}$), constraints $\mathcal{C} = (\mathscr{G}, \prec)$ consisting of a family of k grouping constraints $\mathscr{G}$ and a relation $\prec$ consisting of r ordering constraints, a label height $h > 0$, and a computable function $f : \mathcal{S} \times \mathcal{P} \to \mathbb{R}_0^+$.*

Task: *Find a feasible b-sided po-labeling $\mathcal{L}$ on the candidate ports $\mathcal{P}$ with labels of height h that respects $\mathcal{C}$ and minimizes $f(\mathcal{L})$.*

From now on, we implicitly assume that a feasible labeling also respects the constraints unless stated otherwise.

In the next chapter, we will present an efficient, i.e., polynomial-time, dynamic programming algorithm to solve the 1-Sided Constrained Boundary Labeling problem. However, for the algorithm to be polynomial, we must work with fixed reference points or uniform-height labels. If we have neither of those two, the problem is weakly NP-hard, as we will show in Theorem 3.2. There, we reduce from the Partition problem.

Problem 4 (Partition [GJ79])**.**

Given: *A set $\mathcal{A}$ of N elements and a computable function $w : \mathcal{A} \to \mathbb{Z}^+$.*

Question: *Does there exist a partition of $\mathcal{A}$ into two sets, $\mathcal{A}_1$ and $\mathcal{A}_2$, such that $\sum_{a \in \mathcal{A}_1} w(a) = \sum_{a \in \mathcal{A}_2} w(a)$ holds?*

Theorem 3.1 ([GJ79])**.** Partition *is weakly NP-hard.*

Note that 2-Sided Boundary Labeling, i.e., without any constraints, is weakly NP-hard for non-uniform height labels due to a reduction from Partition [BKSW07].

Theorem 3.2. *Deciding whether an instance of* 1-Sided Constrained Boundary Labeling *possesses a feasible labeling for non-uniform-height labels with sliding reference points is weakly NP-hard, even if we only allow a constant number of grouping constraints.*

Proof. The proposed reduction is conceptually almost identical to the one by Fink and Suri [FS16] to show weak NP-hardness of the boundary labeling problem in the presence of obstacles: We reproduce the obstacle using our constraints. Note that, for ease of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

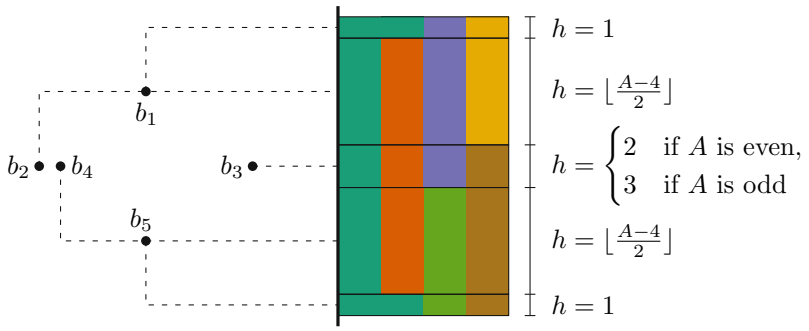


Figure 3.7: The block structure. Labels that contain a slice of the same color are in the same group.

presentation, the sites of the resulting instance are not in general position. However, we can perturb them slightly without affecting the correctness of the construction. Furthermore, any leader-site overlap in the figures can be resolved by similar arguments.

Let $(\mathcal{A}, w)$ be an instance of Partition and assume without loss of generality that we have $\sum_{a \in \mathcal{A}} w(a) = 2A$.[2] We create for each element $a_i \in \mathcal{A}$, $1 \leq i \leq N$, a site s_i whose corresponding label has a height of $w(a_i)$ and place the sites on a horizontal line next to each other. Furthermore, we create five sites, b_1 to b_5, with corresponding labels of height 1 for b_1 and b_5, height $\lfloor \frac{A-4}{2} \rfloor$ for b_2 and b_4, and height 2 or 3 for b_3, depending on whether A is even or odd, respectively. Observe that the height of the labels for b_1 to b_5 sums up to A. We create, in addition, the grouping constraints $\{\{b_1, b_2, b_3, b_4, b_5\}, \{b_1, b_2, b_3\}, \{b_3, b_4, b_5\}, \{b_2, b_3, b_4\}, \{b_1, b_2\}, \{b_4, b_5\}\}$. The sites are placed as shown in Figure 3.7. Above-created grouping constraints enforce that in any feasible labeling, the sites b_1 to b_5 are labeled as indicated in Figure 3.7. Since there is neither an alternative order of the labels nor room to slide around, the labels of these sites must be placed continuously, without any free space, and at that fixed position. Hence, we call the resulting structure a *block*. We create two similar blocks above (using the sites x_1 to x_5) and below (using the sites y_1 to y_5) the sites for $\mathcal{A}$. However, we ensure that we leave between the labels for x_5 and b_1 and the labels for b_5 and y_1 a space of A, respectively. Figure 3.8 visualizes the final layout. The resulting instance $\mathcal{I}(\mathcal{A})$ has $N + 15$ sites and 18 grouping constraints, thus a size polynomial in N. We can also create it in time polynomial in the input size.

Regarding the correctness of the reduction we observe the following.

($\Rightarrow$) Let $(\mathcal{A}, w)$ be a positive instance of Partition and $\mathcal{A}_1$ and $\mathcal{A}_2$ a corresponding solution. We transform this information now into a feasible labeling for $\mathcal{I}(\mathcal{A})$. The blockers are labeled as in Figures 3.7 and 3.8. To determine the position of the labels for the remaining sites, s_1 to s_N, we traverse them from left to right. For any site s_i, $1 \leq i \leq N$, we check whether its corresponding element a_i is in $\mathcal{A}_1$ or $\mathcal{A}_2$. If it is in $\mathcal{A}_1$,

[2] Otherwise, $(\mathcal{A}, w)$ would be a trivial negative instance.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

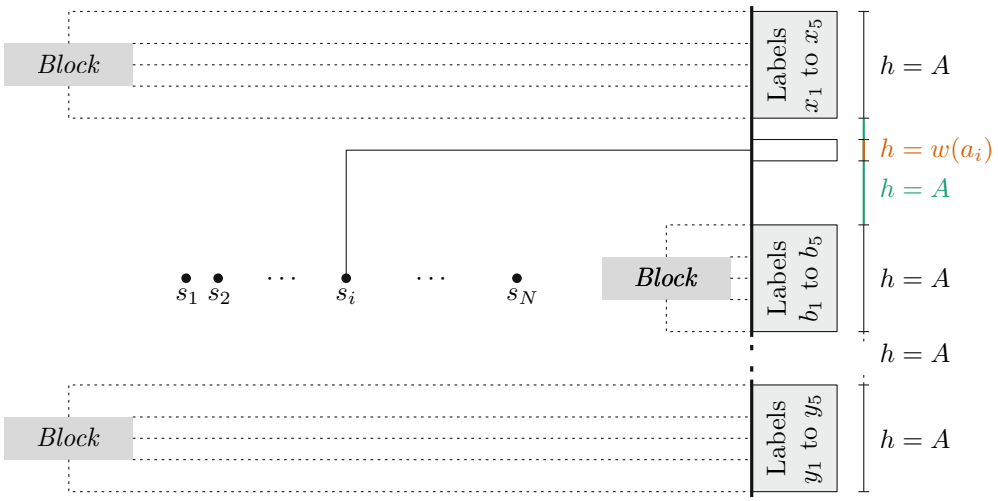


Figure 3.8: The placement of the sites in the reduction. Gray boxes visualize the blocks.

we place the label for s_i as far up as possible, i.e., inside the upper free space of Figure 3.8. On the other hand, if it is in $\mathcal{A}_2$, we place the label for s_i as far down as possible, i.e., inside the lower free space of Figure 3.8. As the sum of the entries of $\mathcal{A}_1$ and $\mathcal{A}_2$ evaluates to A, respectively, and we reflect the values of the sum in the height of the respective labels, it is guaranteed that we can place all labels for sites that represent entries in $\mathcal{A}_1$ in the A-high window between the label for x_5 and the one for b_1. Analogously, the sites that represent entries in $\mathcal{A}_2$ can be placed between the labels for b_5 and y_1. Finally, by traversing the sites from left to right and assigning the outermost possible position for the respective label, we guarantee that the resulting labeling is feasible. Thus, it witnesses the existence of a feasible labeling for $\mathcal{I}(\mathcal{A})$.

($\Leftarrow$) Let $\mathcal{L}$ be a feasible labeling for $\mathcal{I}(\mathcal{A})$. We define $\mathcal{A}_1$ as the set of elements whose corresponding site is labeled between x_5 and b_1 and $\mathcal{A}_2$ as the one where the label is between b_5 and y_1. Recall that in each of these windows, there is only space for that many sites such that the sum of their label heights equals A. Therefore, we know that $\sum_{a\in\mathcal{A}_1} w(a) = \sum_{a\in\mathcal{A}_2} w(a)$ holds, and this partition witnesses that $(\mathcal{A}, w)$ is a positive instance of Partition. □

Observe that in the above reduction, we used grouping constraints to keep the labels for the sites of the blocks in place. We can achieve the same behavior by exchanging them with the ordering constraints $\{b_1 \prec b_2, b_2 \prec b_3, b_3 \prec b_4, b_4 \prec b_5\}$. Of course, we have to perform similar substitutions for x_1 to x_5 and y_1 to y_5. Since we exchange 18 grouping constraints with twelve ordering constraints, we get the following corollary.

Corollary 3.2. *Deciding whether an instance of* 1-Sided Constrained Boundary Labeling *possesses a feasible labeling for non-uniform-height labels with sliding reference points is weakly NP-hard, even if we only allow a constant number of ordering constraints.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 4

Algorithms and Computational Complexity Results

Now that the semantics of grouping and ordering constraints are settled and we have seen an efficient way to represent them, we should seek an algorithm that supports them. Although the literature contains a multitude of boundary labeling algorithms, we have already concluded that none of them is tailored to respect semantic constraints while still offering the flexibility we intend to do.

We have already shown that PQ-A-Graphs allow us to efficiently check that the given constraints are respectable by a set of sites. In Section 4.1, we now propose a polynomial-time algorithm that decides for an instance of 1-Sided Constrained Boundary Labeling whether it has a feasible labeling and, if so, outputs the best one according to an optimization function f. The natural next step would then be to extend this result to 2-Sided Constrained Boundary Labeling. Unfortunately, it turns out that this problem is NP-hard, even for uniform height labels. Section 4.2 is devoted to this result.

4.1 A Polynomial-Time Algorithm for 1-Sided Constrained Boundary Labeling

We begin by stating some key observations in Section 4.1.1 and continue with useful subroutines in Section 4.1.2 before we propose a dynamic programming (DP) algorithm in Section 4.1.3. The algorithm follows the ideas by Benkert et al. [BHKN09]. Throughout this section, we assume that we are given an instance $\mathcal{I} = (\mathcal{S}, \mathcal{P}, \mathcal{C} = (\mathscr{G}, \prec), h, f)$ of 1-Sided Constrained Boundary Labeling that consists of n sites $\mathcal{S}$, m ports $\mathcal{P}$ on the right side of a boundary $\mathcal{B}$ around $\mathcal{S}$, constraints $\mathcal{C} = (\mathscr{G}, \prec)$ consisting of a family of k grouping constraints $\mathscr{G}$ and a relation $\prec$ consisting of r ordering constraints, a label height $h > 0$, and a computable optimization function $f : \mathcal{S} \times \mathcal{P} \to \mathbb{R}_0^+$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

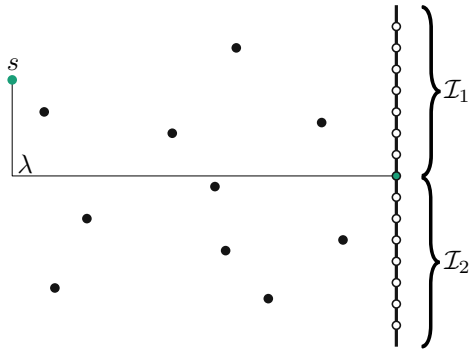


Figure 4.1: The leader λ of the leftmost site s splits an instance into two smaller instances, $\mathcal{I}_1$ and $\mathcal{I}_2$.

4.1.1 Key Observations for Our Dynamic Programming Algorithm

Benkert et al. [BHKN09] already observed that in a planar labeling $\mathcal{L}$ of $\mathcal{I}$, the leader λ of the leftmost site s splits the sites in $\mathcal{L}$ into those that are labeled above λ, and those that are labeled below λ. Since $\mathcal{L}$ is planar, no leader will cross λ, and the two resulting sub-labelings are disjoint and independent. We visualize this again in Figure 4.1. This finding, which we formally state in Observation 4.1, gives rise to use dynamic programming to compute and then combine optimal labelings of sub-instances.

Observation 4.1 ([BHKN09])**.** *In a planar labeling $\mathcal{L}$ of an instance $\mathcal{I}$ of* 1-Sided Constrained Boundary Labeling*, the leader λ connecting the leftmost site $s \in \mathcal{S}$ with some port $p \in \mathcal{P}$ splits the instance into two independent sub-instances, $\mathcal{I}_1$ and $\mathcal{I}_2$, excluding s and p.*

Recall that a *po*-leader λ is fully defined by a site $s \in \mathcal{S}$ and a port $p \in \mathcal{P}$. Together with Observation 4.1, this implies that we can fully describe the geometric structure of an instance $\mathcal{I}$ of 1-Sided Constrained Boundary Labeling using a tuple $I = (s_1, p_1, s_2, p_2)$, for $s_1, s_2 \in \mathcal{S}$ and $p_1, p_2 \in \mathcal{P}$. The corresponding leaders $\lambda_1 = (s_1, p_1)$ and $\lambda_2 = (s_2, p_2)$ bound the instance described by I from above and below, respectively. In the following, we use I to denote the geometric part, i.e., the sites and ports, of an instance that occurs in our DP-Algorithm. By adding artificial sites s_0 and s_{n+1}, and ports p_0 and p_{m+1} that are connected with leaders $\lambda_0 = (s_0, p_0)$ and $\lambda_{n+1} = (s_{n+1}, p_{m+1})$ at the very top and bottom of the boundary, respectively, we can also describe the initial instance by a tuple $I = (s_0, p_0, s_{n+1}, p_{m+1})$. As a site and port combination also describes a leader, we sometimes use the shorthand notation $I = (\lambda_1, \lambda_2)$. A labeling $\mathcal{L}$ *contains* I if λ_1 and λ_2 are present in $\mathcal{L}$, i.e., if we label s_1 at port p_1 and s_2 at port p_2. We denote by $\mathcal{S}(I)$ the sites within the instance defined by I, *excluding* those used in the definition of I. $\mathcal{P}(I)$ and $\prec(I)$ are defined analogously. Observe that Benkert et al. [BHKN09] and Fink and Suri [FS16] described instances with similar tuples.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

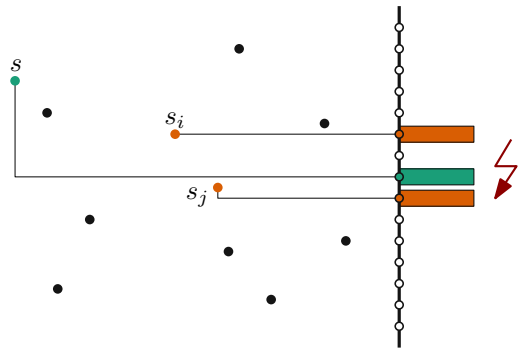


Figure 4.2: The leftmost site s splits the orange group containing s_i and s_j. Thus, this cannot result in a feasible labeling that respects the orange grouping constraint.

While Observation 4.1 also holds without grouping or ordering constraints, Observations 4.2 and 4.3 follow from our definition of grouping and ordering constraints, respectively.

Observation 4.2. *In a feasible labeling $\mathcal{L}$ of an instance $\mathcal{I}$ of* 1-Sided Constrained Boundary Labeling *that respects the grouping constraints $\mathscr{G}$, the leader λ of the leftmost site $s \in \mathcal{S}$ never splits sites $s_i, s_j \in \mathcal{S}$, $1 \leq i, j \leq n$, $i \neq j$, of a group $\mathcal{G} \in \mathscr{G}$ with $s_i, s_j \in \mathcal{G}$ and $s \notin \mathcal{G}$.*

Observation 4.2 follows trivially from the consequences of respecting a grouping constraint. If the leftmost site s would split s_i and s_j that are inside the group $\mathcal{G}$ with $s \notin \mathcal{G}$, then the triple s_i, s_j, and s would be a witness that the resulting labeling $\mathcal{L}$ does not respect the grouping constraints. Figure 4.2 visualizes this observation: The label ℓ for the leftmost site s splits the group containing s_i and s_j. Thus, in any feasible labeling where we place ℓ there, ℓ will separate the orange labels, thus implying an infeasible labeling as in Figure 4.2.

Observation 4.3. *In a feasible labeling $\mathcal{L}$ of an instance $\mathcal{I}$ of* 1-Sided Constrained Boundary Labeling *that respects the ordering constraints $\prec$, the leader λ of the leftmost site $s \in \mathcal{S}$ never splits sites s_i and s_j, $1 \leq i, j \leq n$, $i \neq j$, with s_i above λ and s_j below λ, for which we have $s_j \prec s_i$, $s_j \prec s$, or $s \prec s_i$.*

Observation 4.3 follows readily from the definition of the ordering constraints. Since the leader of s splits the instance in a way such that s_i will be labeled above s, and s above s_j, the resulting labeling will inevitably violate any of the stated constraints.

4.1.2 Subroutines of Our Dynamic Programming Algorithm

Our DP-Algorithm uses several subroutines that work with the PQ-A-Graph $\mathcal{T}$. Each of the following sections introduces one of these subroutines and formally analyzes its running time. To do that properly, we have to introduce the notion of a site being above

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

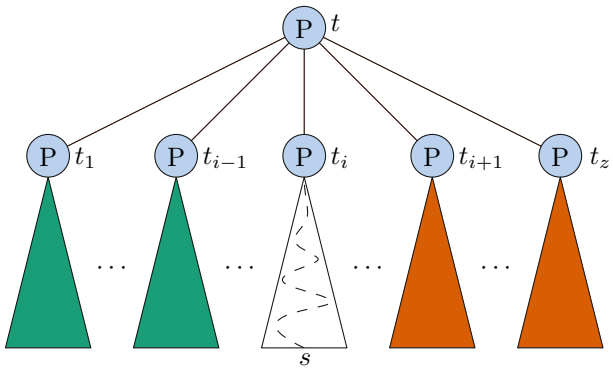


Figure 4.3: We consider the sites in the green subtrees above s at t and those in the orange subtrees below s at t.

or below another site at an internal node of $\mathcal{T}$. Let s be a site and t an ancestor of s in $\mathcal{T}$. Assume that t has the children $t_1, \ldots, t_z$ in this order from left to right. Let T_i, $1 \leq i \leq z$, be the subtree that contains the site s, rooted at t_i. The labels for all the sites contained in the subtrees $T_1, \ldots, T_{i-1}$, i.e., the sites in $\bigcup_{j=1}^{i-1} \text{leaves}(T_j)$, will be placed above the label for s in any labeling $\mathcal{L}$ of $\mathcal{S}$ in which the sites in T are ordered according to $t_1, \ldots, t_z$. Therefore, we call the sites in $\bigcup_{j=1}^{i-1} \text{leaves}(T_j)$ *above* s (at t). Analogously, the sites in the subtrees $T_{i+1}, \ldots, T_z$, i.e., represented by $\bigcup_{j=i+1}^{z} \text{leaves}(T_j)$, are the sites *below* s (at t). Note that the sites represented by $\text{leaves}(T_i)$ fall in neither of those two sets. However, they are above or below s at an earlier (i.e., lower) ancestor t' of s. Figure 4.3 visualizes this. Each internal node t and t_1 to t_z in Figure 4.3 could also be a Q-node instead of a P-node. Note that while the notion of a site being above or below s at t talks about the position of the labels according to an order of the children of a node in $\mathcal{T}$, being left or right of a site s still talks about the (geometric) position of the sites to each other.

Verifying that the Label for a Site Respects the Constraints

The first subroutine deals with the question of whether, for a given instance I in our DP-Algorithm, the label ℓ for the leftmost site $s \in \mathcal{S}(I)$ would respect the constraints if placed at the candidate port $p \in \mathcal{P}(I)$. Throughout this section, we let $\mathcal{T}$ denote the subtree of the PQ-A-Graph rooted at $\text{lca}(s_1, s_2)$. Note that $\mathcal{T}$ represents the (inclusion-wise) smallest canonical group induced by the original PQ-A-Graph that contains all the sites in $\mathcal{S}(I)$, together with s_1 and s_2. We denote with t_r the root of $\mathcal{T}$.

Let t_l be the leaf for s in $\mathcal{T}$. There is a unique path from t_l to t_r in $\mathcal{T}$, which we traverse bottom up and consider each internal node t on it. We say that the (candidate) port p *respects for* s *at* t the constraints imposed by $\mathcal{T}$ if the following holds depending on the type of t. Recall that λ denotes the (candidate) leader for s.

P-node: If t is a P-node, there must exist at least one permutation π of the children of t in which *all* the sites in $\mathcal{S}$(I) above s at t (in the permutation π) are above λ, and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

all the sites in $\mathcal{S}$(I) that will be in π below s at t are below λ. This means that it cannot be the case that some sites of the same subtree T are above λ, while others are below λ. Furthermore, in this permutation π, we require that there is no pair of sites s_i, s_j, $1 \leq i, j \leq n$, $i \neq j$, such that s_i is in the subtree T_x, and s_j is in the subtree T_y, $1 \leq x, y \leq z$, $x \neq y$, and we have $\pi(t_x) < \pi(t_y)$, but also the arc $(s_j, s_i) \in \mathcal{T}$. Let $t_1, \ldots, t_z$ be the children of t and t_i, $1 \leq i \leq z$, the child that contains the site s. To check whether a desired permutation π exists, we partition the children of t, except t_i, into two sets, t_{above} and t_{below}. In the following, we differentiate whether t is an internal node (of $\mathcal{T}$), an internal node on the path from s_1 (or s_2) to the root t_r of $\mathcal{T}$, or the root of $\mathcal{T}$. If t is an internal node on the path, we only consider the case where it is on the path from s_1 to t_r, as we can derive the other case by exchanging *above* with *below*.

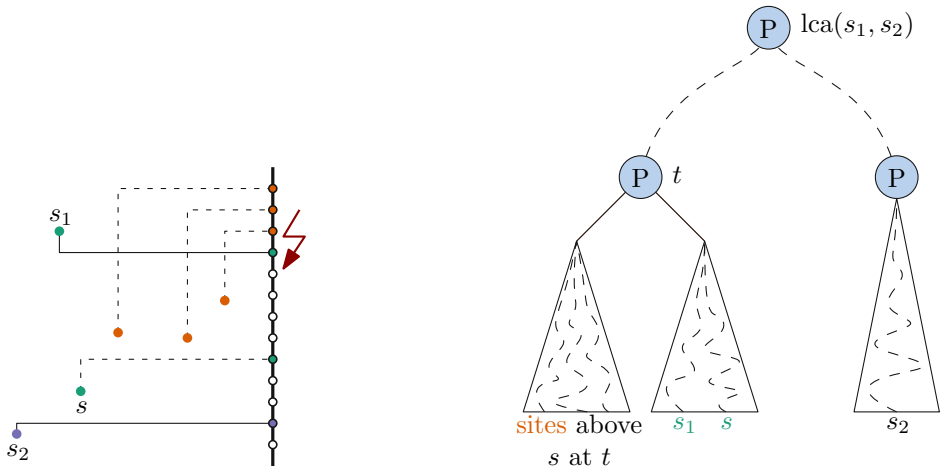
If t is an internal node *not* on the path from s_1 to t_r, then all children of t contain only sites within the instance I. For a child t_j, $1 \leq j \leq z$, $i \neq j$, we check whether all the sites in the subtree T_j rooted at t_j are above λ, or if all are below λ. In the former case, we put t_j in the set t_{above}, and in the latter case in t_{below}. Note that if neither of these cases applies, we know that λ would split a (canonical) group to which s does not belong, and hence, p does not respect the constraints at t for s, and we can return with failure.

If t is an internal node on the path from s_1 to t_r, we proceed as before. However, if a child t_j, $1 \leq j \leq z$, $i \neq j$, contains only sites outside the instance, we immediately put it in t_{above}. This follows from the definition of I, as these sites are outside the instance and must, therefore, be above s at t. If t_j contains $s_1 \in \text{leaves}(T_j)$, it can contain some sites in I and others outside I. We only check whether all other sites in t_j that are in I, i.e., all the sites in $\text{leaves}(T_j) \cap \mathcal{S}(I)$, are above λ. Similar to before, the definition of I already enforces that the sites must be above s at t, as s_1 is above s at t. Hence, we can return with failure if these checks do not succeed. If $s_1 \in \text{leaves}(T_i)$ holds, then t_{above} must not contain a child t_j containing sites from I, as they would then be labeled outside I, violating the definition of I. Figure 4.4 visualizes this.

Finally, if $t = t_r$ holds, we must be more careful, as sites outside the instance can now be above or below s at t. If for a child t_j, $1 \leq j \leq z$, $i \neq j$, all sites are inside the instance we proceed as usual. Similarly, if T_j contains s_1 or s_2, we can put t_j in t_{above} or t_{below}, respectively. If there are also sites inside I, we first have to ensure that this complies with the position of the leader λ and otherwise return with failure. Observe that this guarantees that s_1 will be above s and s_2 below s at t as required by the definition of I. If T_j does not contain the sites s_1 and s_2, and only sites outside I, then we ignore t_j and put it in neither of the sets. As t is a P-node, t_j can be placed either above or below s at t. Since T_j does only contain sites outside I and does not include s_1 or s_2, it is unclear, unless there is a suited ordering or grouping constraint, whether to put t_j in t_{above} or t_{below}. However, at the latest when we placed the leaders for s_1 or s_2, we ensured that we did not violate a constraint that involves sites of T_j. Hence, it is safe to ignore this child. Similar to before, if $s_1 \in \text{leaves}(T_i)$ holds, then t_{above} must not contain a t_j having sites from I. The same holds with t_{below} if we have $s_2 \in \text{leaves}(T_i)$.

Observe that we query the position of each site s' in a T_j, $1 \leq j \leq z$, $i \neq j$, $\mathcal{O}(1)$ times

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) If s and s_1 are in the same group, and the orange sites are above s (and s_1) at t, then the resulting labeling, indicated with the dashed lines, is infeasible.

(b) Parts of the PQ-A-Graph $\mathcal{T}$ for the instance from Figure 4.4a.

Figure 4.4: Visualization of the arguments why t_{above} must, in some cases, not contain subtrees with sites from the instance.

and compare it each time to λ, which takes constant time, or check whether it is in the instance, which takes also constant time. Afterwards, we do not consider this site anymore. Therefore, this process can be implemented in $\mathcal{O}(|T_j|) = \mathcal{O}(n)$ time.

Having t_{above} and t_{below}, we check if there exists a site s_u, represented by a leaf $l_u \in \bigcup_{t_u \in t_{\text{above}}} \text{leaves}(T_u)$, a site s_v, represented by a leaf $l_v \in \text{leaves}(T_i)$, and a site s_w, represented by a leaf $l_w \in \bigcup_{t_w \in t_{\text{below}}} \text{leaves}(T_w)$, such that we have the arc (s_v, s_u), (s_w, s_u), or (s_w, s_v) in $\mathcal{T}$. If so, we know that we violate an ordering constraint and return with failure. To perform these checks efficiently, we can maintain, while computing t_{above} and t_{below}, a look-up table that stores for each site whether it belongs to t_{above}, t_{below}, or T_i. Then we check for each of the $r_I = |\prec(I)|$ many arcs on the leaves of $\mathcal{T}$ in constant time whether we violate it or not. Consequently, we can implement these checks to run in $\mathcal{O}(n + r_I)$ time, which already includes the time required to compute the look-up tables.

Q-node: If t is a Q-node, one of the following two cases must hold.

Either all sites s' above s at t are above λ. This can be sites in the instance I, for which we can check the position with respect to λ, or outside the instance but in $\mathcal{T}$, for which we know where they must be due to the definition of I. Similarly, all sites below s at t are below λ. Hence, we keep the order of the children at the node t as they are. Furthermore, there must not be an arc $(s_u, s_v) \in \mathcal{T}$, $1 \leq u, v \leq n$, that prevents this ordering.

Or all sites above s at t are below λ, and all sites below s at t are above λ. This means that we need to inverse the order of the children at the node t. Furthermore, there must

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

not be an arc $(s_u, s_v) \in \mathcal{T}$, $1 \leq u, v \leq n$, that prevents this inversion of the ordering.

Similar to P-nodes, we differentiate whether t is an internal node of $\mathcal{T}$, an internal node on the path from s_1 or s_2 to the root t_r, or the root t_r of $\mathcal{T}$. Observe that we use the arguments we already gave for P-nodes to justify why it is sometimes enforced, by the definition of I, where a child of t must be. Furthermore, note that having s_1 or s_2 in some T_j, $1 \leq j \leq z$, $i \neq j$, predetermines, by the definition of I, which of the two allowed inversions we will end up with. Therefore, in such a case, we only need to ensure that the leader λ respects this. Altogether, these observations guarantee that we do not contradict the definition of I. The checks for a Q-node can be implemented, by the same arguments as for a P-node, to run in $\mathcal{O}(n + r_I)$ time, where r_I denotes the number of arcs in $\mathcal{T}$.

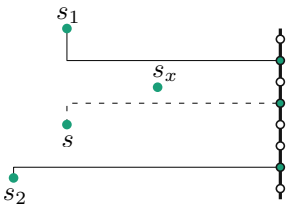
The (candidate) port p *respects the constraints for* s imposed by $\mathcal{T}$ in the instance $I = (s_1, p_1, s_2, p_2)$ if it respects them for s at every node t on the path from s to the root of $\mathcal{T}$. Let RespectsConstraints$(I, \mathcal{T}, \lambda)$ be a procedure that implements above steps and returns "true" if they succeed and "false" otherwise, i.e, checks whether the leader λ for s would respect the constraints imposed by $\mathcal{T}$ in instance I.

Lemma 4.1. *Let I be an instance of our DP-Algorithm with the constraints expressed by $\mathcal{T}$ and $p \in \mathcal{P}(I)$. Checking whether the leader $\lambda = (s, p)$ for the leftmost site s in $\mathcal{S}(I)$ respects the constraints using* RespectsConstraints$(I, \mathcal{T}, \lambda)$ *takes $\mathcal{O}(n^2 + nr)$ time.*

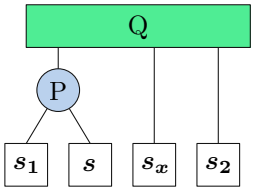
Proof. When describing the procedure RespectsConstraints$(I, \mathcal{T}, \lambda)$, we have already observed that we can check whether the port p respects for s the constraints imposed by $\mathcal{T}$ at a node t on the path from s to $\text{lca}(s_1, s_2)$ in time $\mathcal{O}(n + r_I)$, where r_I denotes the number of ordering constraints, i.e., arcs, in $\mathcal{T}$. Observe that r_I is at most r. Hence, $\mathcal{O}(n + r_I) = \mathcal{O}(n + r)$. Since we have to check this on each of the $\mathcal{O}(n)$ nodes t on the path to the $\text{lca}(s_1, s_2)$, we end up with a running time of $\mathcal{O}(n^2 + nr)$. □

Finally, we want to argue why we only have to check the subtree of the PQ-A-Graph rooted at the $\text{lca}(s_1, s_2)$, where s_1 and s_2 define the boundary of the instance. This is because we must respect any constraint that originates from a node t further up in the tree: All sites from $\mathcal{S}$(I) are in the subtree of the same child of t. In particular, observe that the subtree we considered for the instance I' that contains I has its root $t_{r'}$ at t_r or above t_r, i.e., $t_{r'} = t_r$ or $t_{r'}$ is an ancestor of t_r. Thus, on the way from the leftmost site in I' to $t_{r'}$, we "passed by" t_r and ensured that we could respect the constraints. We still want to point out why we must consider all the internal nodes on the way to the root of $\mathcal{T}$. To show this, we give an example in Figure 4.5 where we cannot stop earlier. The constraint that includes s_1 is vacuously fulfilled, as there is no further site in this subtree (compare Figure 4.5a with Figure 4.5b). However, if we would now connect s to the port indicated with the dashed leader, we would not recognize that we violate the order imposed by the Q-node. Since we recognize that this constraint is violated only at the $\text{lca}(s_1, s_2)$, i.e., the root of the subtree, we can conclude that we have to check for all internal nodes up to and including the root of the subtree of $\mathcal{T}$ rooted at $\text{lca}(s_1, s_2)$ whether the constraints are satisfied.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) An instance where we have to check the constraints up to the $\text{lca}(s_1, s_2)$ in $\mathcal{T}$.



(b) Parts of the PQ-A-Graph $\mathcal{T}$ for the instance from Figure 4.5a.

Figure 4.5: A (sub-)instance, together with the subtree $\mathcal{T}$ rooted at $\text{lca}(s_1, s_2)$, showing that we cannot stop checking the constraints earlier.

Verifying that a Label Candidate is Feasible for a Site

As we will see soon, our DP-Algorithm makes use of Observation 4.1 by testing, in an instance I of our DP-Algorithm, for the leftmost site $s \in \mathcal{S}(I)$ which of the candidate ports $p \in \mathcal{P}(I)$ can yield a feasible labeling $\mathcal{L}$. To do that, we make use of the procedure Feasible$(I, \mathcal{T}, p)$, which checks whether the port p is feasible for s with respect to the constraints expressed by $\mathcal{T}$. Formally, we say that a (candidate) port $p \in \mathcal{P}(I)$ is *feasible* for the leftmost site $s \in \mathcal{S}(I)$ given $\mathcal{T}$ if all of the following constraints are satisfied. Note that ℓ denotes the label for the site s and $\lambda = (s, p)$ the corresponding leader.

(a) The label ℓ does not overlap with the labels ℓ_1, placed at p_1, and ℓ_2, placed at p_2, for the sites s_1 and s_2, respectively, that define the instance I.

(b) The leader λ does not intersect with a site $s' \in \mathcal{S}(I)$, $s' \neq s$.

(c) In both resulting sub-instances $I_1 = (s_1, p_1, s, p)$ and $I_2 = (s, p, s_2, p_2)$, there are enough ports for all sites, i.e., $|\mathcal{S}(I_i)| \leq |\mathcal{P}(I_i)|$, for $i = 1, 2$.

(d) RespectsConstraints$(I, \mathcal{T}, \lambda)$ must yield true.

Lemma 4.2. *Let I be an instance of our DP-Algorithm with the constraints expressed by $\mathcal{T}$. We can check whether the port $p \in \mathcal{P}(I)$ is feasible for the leftmost site $s \in \mathcal{S}(I)$ using* Feasible$(I, \mathcal{T}, p)$ *in $\mathcal{O}(n^2 + nr + \log m)$ time.*

Proof. To prove the claimed running time, we will use the results of Lemma 4.1. We continue arguing the running time for each of the above constraints individually.

Constraint (a). In Constraint (a), we must check whether two labels overlap, which we can do in constant time. Observe that by our assumption that all labels are of height h, and arbitrary width, together with our requirement of fixed ports, namely in the center of the label, we only have to check whether $y(p_2) + h \leq y_p \leq y(p_1) - h$ holds.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Constraint (b). For Constraint (b), we can check for each site $s' \in \mathcal{S}(I)$, $s' \neq s$, to the right of s, i.e., each site where we have $x(s) < x(s')$, that it does not have the same y-coordinate as the port p, i.e, it cannot hold $y(s') = y(p)$. This takes $\mathcal{O}(n)$ time.

Constraint (c). To check this constraint efficiently, we assume that we can access a range tree storing the sites, and a sorted list containing the ports. We will account for this when we discuss the overall properties of our DP-Algorithm. With this assumption, we can compute the number of ports in an instance by running two binary searches for p_1 and p_2 that define the instance, which takes $\mathcal{O}(\log m)$ time. To count the number of sites in the instance, we can run a counting range query on the sites, which takes $\mathcal{O}(\log n)$ time [BCKO08]. In the end, we only have to compare the retrieved numbers. Combining this, we arrive at a running time of $\mathcal{O}(\log n + \log m)$.

Constraint (d). Checking whether RespectsConstraints$(I, \mathcal{T}, \mathcal{L})$ yields true can be done in $\mathcal{O}(n^2 + nr)$ time using the results from Lemma 4.1.

Combining all, we get a running time of $\mathcal{O}(n^2 + nr + \log m)$. □

4.1.3 Putting It Together: Our Dynamic Programming Algorithm

Now that we have discussed in Section 4.1.2 the subroutines that serve as the ingredients of our DP-Algorithm, it is time to put them together in the right form. We remind the reader that our DP-Algorihm is based on the approach by Benkert et al. [BHKN09] and heavily uses the observations presented in Section 4.1.1.

Let D be a DP-Table of size $\mathcal{O}(n^2m^2)$ that keeps track of the optimal solutions of instances that occur in our DP-Algorithm. For an instance $I = (s_1, p_1, s_2, p_2)$, we denote with $D[(s_1, p_1, s_2, p_2)] = f(\mathcal{L}^*)$ the cost of a feasible labeling $\mathcal{L}^*$ of I that is optimal for the optimization function f. If the instance I does not possess a feasible labeling, we set $D[(s_1, p_1, s_2, p_2)] = \infty$. We will use $D[I]$ as a shorthand for $D[(s_1, p_1, s_2, p_2)]$ if $I = (s_1, p_1, s_2, p_2)$ is clear from the context.

Having all notions, we can state the recurrence relation of our DP-Algorithm. If I does not contain a site, i.e., if $\mathcal{S}(I) = \emptyset$ holds, we set $D[I] = 0$. Otherwise, let s be the leftmost site in the instance. For an instance $I = (s_1, p_1, s_2, p_2)$, with $\lambda_1 = (s_1, p_1)$ and $\lambda_2 = (s_2, p_2)$, we use the following relation.

$$D[I] = \min_{\substack{p \in \mathcal{P}(I) \text{ where} \\ \textsc{Feasible}(I, \mathcal{T}, p) \text{ is true}}} (D[(\lambda_1, \lambda)] + D[(\lambda, \lambda_2)]) + f(\lambda), \tag{4.1}$$

where $\lambda = (s, p)$ is the corresponding leader if we label s at the port p. We assume that the minimum of the empty set is ∞, i.e., if we have $\mathcal{P}(I) = \emptyset$ or if for all $p \in \mathcal{P}(I)$ the subroutine Feasible$(I, \mathcal{T}, p)$ yields false, then we have $D[I] = \infty$.

The correctness of the recurrence relation from Equation (4.1) follows by combining Observations 4.1 to 4.3. Since the leader of the leftmost site splits an instance into two

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

independent smaller sub-instances (Observation 4.1), if there exists a feasible labeling $\mathcal{L}$ of an instance $\mathcal{I}$ of the 1-Sided Constrained Boundary Labeling problem, then we can find it by trying out for the leftmost site s in an instance I that occurs in the DP-Algorithm all possible ports p of the instance. As not all ports $p \in \mathcal{P}(I)$ might lead to a feasible labeling (Observations 4.2 and 4.3), we must ensure that a port p does not violate a constraint, i.e., that Feasible$(I, \mathcal{T}, p)$ yields true. Since the cost of a labeling for an instance I is composed of the costs for the two sub-instances together with the cost of the leader that led to the creation of these sub-instances, we conclude that the recurrence relation of Equation (4.1) computes indeed the cost of the best feasible labeling $\mathcal{L}^*$ for $\mathcal{I}$ according to the optimization function f. Recall that by adding two auxiliary sites s_0, s_{n+1} and ports p_0, p_{m+1}, the entry $D[s_0, p_0, s_{n+1}, p_{m+1}]$ will store in the end $f(\mathcal{L}^*)$, or ∞, if $\mathcal{I}$ does not possess a feasible labeling.

We continue with showing the running time of our DP-Algorithm. Recall that our algorithm builds on two subroutines whose individual running times we discussed in Lemmas 4.1 and 4.2.

Theorem 4.1. *Let $\mathcal{I} = (\mathcal{S}, \mathcal{P}, \mathcal{C} = (\mathscr{G}, \prec), h, f)$ be an instance of the* 1-Sided Constrained Boundary Labeling *problem with $n = |\mathcal{S}|$, $m = |\mathcal{P}|$, $k = |\mathscr{G}|$, and $r = |\prec|$. We can compute $f(\mathcal{L}^*)$ for a feasible labeling $\mathcal{L}^*$ of $\mathcal{I}$ that minimizes f in $\mathcal{O}(n^5 m^3 \log m + k + \|\mathscr{G}\| + nmf(\cdot,\cdot))$ time using $\mathcal{O}(n^2 m^2)$ space, or conclude that no such labeling exists, where $\mathcal{O}(f(\cdot,\cdot))$ denotes the evaluation time of $f(s,p)$, for arbitrary $s \in \mathcal{S}$ and $p \in \mathcal{P}$, i.e., the evaluation complexity of the optimization function f.*

Proof. Let $\mathcal{I}$ be an instance of 1-Sided Constrained Boundary Labeling with the constraints $\mathcal{C} = (\mathscr{G}, \prec)$. From Lemma 3.4, we know that we can check in $\mathcal{O}(n+k+r+\|\mathscr{G}\|)$ time whether the constraints $\mathcal{C}$ are respectable for $\mathcal{S}$. For the sake of the proof, let us assume that they are, as otherwise $\mathcal{I}$ does not possess a feasible labeling $\mathcal{L}$. Therefore, we can obtain, in this time, the corresponding PQ-A-Graph $\mathcal{T}$ that uses $\mathcal{O}(n + r)$ space.

As further preprocessing steps, we create an additional list storing the ports $p \in \mathcal{P}$ sorted by $y(p)$ in ascending order and a range tree on the sites. We can do the former in $\mathcal{O}(m \log m)$ time and $\mathcal{O}(m)$ space, and the latter in $\mathcal{O}(n \log n)$ time and space [BCKO08]. In addition, we compute for each internal node t of the PQ-A-Graph $\mathcal{T}$ the canonical group induced by the subtree rooted at t, and for each pair of sites s_1 and s_2 the lca(s_1, s_2) in $\mathcal{T}$. We can do the former in $\mathcal{O}(n^2)$ time and space by traversing $\mathcal{T}$ bottom-up, as $\mathcal{T}$ has $\mathcal{O}(n)$ internal nodes and each canonical group is of size at most n. We can do the latter in $\mathcal{O}(n^2)$ time [BF00]. Furthermore, we create a look-up table for $f(s,p)$, i.e., we compute $f(s,p)$ for all $(s,p) \in \mathcal{S} \times \mathcal{P}$. Creating this look-up table consumes $\mathcal{O}(nm)$ space and requires $\mathcal{O}(nmf(\cdot,\cdot))$ time, where $\mathcal{O}(f(\cdot,\cdot))$ states the time required to compute $f(s,p)$ for arbitrary $s \in \mathcal{S}$ and $p \in \mathcal{P}$. By adding artificial sites s_0 and s_{n+1}, and ports p_0 and p_{m+1}, we can describe any instance by a tuple $I = (s_1, p_1, s_2, p_2)$, and in particular $\mathcal{I}$ by $I = (s_0, p_0, s_{n+1}, p_{m+1})$.

We then continue filling the table D top-down using memoization. This guarantees that we have to evaluate each sub-instance I at most once and only those that arise

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

from feasible (candidate) leaders. For each such I, we have to, when evaluating the relation from Equation (4.1), check for $\mathcal{O}(m)$ candidate ports p if they are feasible for the leftmost site s. We can obtain s in $\mathcal{O}(n)$ time. For each candidate port p, we first have to check whether it is feasible for s, which we can do in $\mathcal{O}(n^2 + nr + \log m)$ time due to Lemma 4.2. Since we have pre-computed all values for $f(s, p)$, this is the overall time required for a single candidate port p. Hence, evaluating all candidate ports p takes $\mathcal{O}(m\,(n^2 + nr + \log m))$ time. Two sites and two ports describe an instance. Therefore, there are $\mathcal{O}(n^2m^2)$ possible instances I that we have to evaluate in the worst case. Hence, our DP-algorithm solves the 1-Sided Constrained Boundary Labeling problem, for a given instance $\mathcal{I}$, in $\mathcal{O}(n^2m^2(n + m\,(n^2 + nr + \log m)) + k + \|\mathscr{G}\| + nmf(\cdot,\cdot)) = \mathcal{O}(n^4m^3 + n^3m^3r + n^2m^3 \log m + k + \|\mathscr{G}\| + nmf(\cdot,\cdot))$ time using $\mathcal{O}(n^2m^2)$ space. Note that the above bounds dominate the time and space required for the remaining preprocessing steps. Observe that $r = \mathcal{O}(n^2)$ holds. Hence, if r is small, i.e., $r = \mathcal{O}(n)$, above bounds can be simplified to $\mathcal{O}(n^4m^3 + n^2m^3 \log m + k + \|\mathscr{G}\| + nmf(\cdot,\cdot))$. On the other hand, if r is large, i.e., $r = \Theta(n^2)$, above bounds yield $\mathcal{O}(n^5m^3 + n^2m^3 \log m + k + \|\mathscr{G}\| + nmf(\cdot,\cdot))$. Although yielding a higher bound, to ease readability, we will combine above observations into $\mathcal{O}(n^5m^3 \log m + k + \|\mathscr{G}\| + nmf(\cdot,\cdot))$, resulting in the claimed bounds. □

We end our discussion of the DP-Algorithm with two observations. First, we observe that we can store in an auxiliary table D' for each instance $I = (s_1, p_1, s_2, p_2)$ the port $p \in \mathcal{P}(\mathcal{I})$ that yielded the minimum cost, i.e.,

$$D'[I] = \arg \min_{\substack{p \in \mathcal{P}(I) \text{ where} \\ \textsc{Feasible}(I,\mathcal{T},p) \text{ is true}}} (D[(\lambda_1, \lambda)] + D[(\lambda, \lambda_2)]) + f(\lambda).$$

As D' has the same space requirement as our main DP-table D, i.e., $\mathcal{O}(n^2m^2)$, we arrive at Corollary 4.1.

Corollary 4.1. *Let $\mathcal{I} = (\mathcal{S}, \mathcal{P}, \mathcal{C} = (\mathscr{G}, \prec), h, f)$ be an instance of the* 1-Sided Constrained Boundary Labeling *problem with $n = |\mathcal{S}|$, $m = |\mathcal{P}|$, $k = |\mathscr{G}|$, and $r = |\prec|$. We can compute a feasible labeling $\mathcal{L}^*$ of $\mathcal{I}$ that minimizes f in $\mathcal{O}(n^5m^3 \log m + k + \|\mathscr{G}\| + nmf(\cdot,\cdot))$ time using $\mathcal{O}(n^2m^2)$ space, if such a labeling exists.*

Second, if we ask for a leader-length- or leader-bend-minimal labeling or just want to compute some feasible labeling, we have defined in Section 2.4.1 the corresponding functions f. As we can evaluate them in constant time, i.e., we have $\mathcal{O}(f(\cdot,\cdot)) = \mathcal{O}(1)$, we get Corollary 4.2.

Corollary 4.2. *Let $\mathcal{I} = (\mathcal{S}, \mathcal{P}, \mathcal{C} = (\mathscr{G}, \prec), h, f)$ be an instance of the* 1-Sided Constrained Boundary Labeling *problem with $n = |\mathcal{S}|$, $m = |\mathcal{P}|$, $k = |\mathscr{G}|$, and $r = |\prec|$. We can compute a feasible/leader-length-minimal/leader-bend-minimal labeling $\mathcal{L}^*$ of $\mathcal{I}$ in $\mathcal{O}(n^5m^3 \log m + k + \|\mathscr{G}\|)$ time using $\mathcal{O}(n^2m^2)$ space, if such a labeling exists.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.2 The Computational Complexity of 2-Sided Constrained Boundary Labeling

In contrast to results for the general 2-Sided Boundary Labeling problem, where we have polynomial-time algorithms for uniform-height labels [BKSW07; BHKN09] and proofs showing weakly NP-hardness for non-uniform-height labels [BKSW07], we will show in this section that already the problem of finding some feasible two-sided boundary labeling in the presence of grouping and ordering constraints is NP-hard, even for uniform-height labels. We define, based on the definition of 2-Sided Constrained Boundary Labeling, the problem Existence of 2-Sided Constrained Boundary Labeling.

Problem 5 (Existence of 2-Sided Constrained Boundary Labeling)**.**

Given: *A set of n sites $\mathcal{S} = \{s_1, \ldots, s_n\}$, a set of m ports $\mathcal{P} = \{p_1, \ldots, p_m\}$ on two opposite sides of the boundary $\mathcal{B}$ (around $\mathcal{S}$), constraints $\mathcal{C} = (\mathcal{G}, \prec)$ consisting of a family of k grouping constraints $\mathcal{G}$ and a relation $\prec$ consisting of r ordering constraints, and a label height $h > 0$.*

Question: *Does there exist a feasible two-sided po-labeling $\mathcal{L}$ on the ports $\mathcal{P}$ of height h that respects $\mathcal{C}$ with labels?*

Throughout this section, we assume, without loss of generality, $h = 1$. We remind the reader of Assumptions 3 and 4, which we made when introducing the grouping and ordering constraints, respectively. Assumption 3 enforces that we must label sites that belong to the same group on the same side of $\mathcal{B}$. For ordering constraints, Assumption 4 states that the ordering constraints are considered on each side of $\mathcal{B}$ separately and independently. Especially the latter assumption has two immediate implications, which we state in the following observation and visualize in Figure 4.6.

Observation 4.4. *If we have the ordering constraint $s \prec s'$, we can* deactivate *it by labeling s and s' on* different sides *of $\mathcal{B}$ to allow s to be labeled below s'. If we must label s below s', then in any feasible labeling $\mathcal{L}$ that respects this ordering constraint, we* must deactivate *this constraint by labeling s and s' on different sides of $\mathcal{B}$.*

In the hardness proof, we will, in particular, use the latter consequence. We will reduce from Monotone One-In-Three Sat, which is defined as follows.

Problem 6 (Monotone One-In-Three Sat [GJ79])**.**

Given: *A boolean formula φ in conjunctive normal form, where each clause consists of exactly* three different positive *literals.*

Question: *Does there exist a truth assignment that satisfies* exactly one *literal in each clause?*

Theorem 4.2 ([Sch78])**.** Monotone One-In-Three Sat *is NP-complete.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

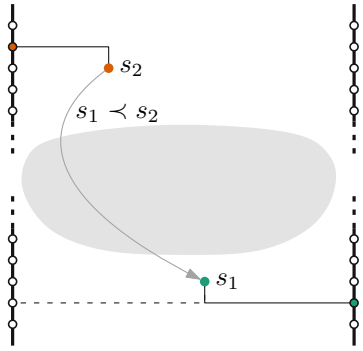


Figure 4.6: If the geometric properties of the instance enforce that s_1 must be labeled below s_2, visualized with the gray area, then we must label them on different sides if we have $s_1 \prec s_2$. If we label them on the same side, as indicated with the dashed leader for s_1, then the ordering constraint will be violated.

4.2.1 Building Blocks Used in the Reduction

Let φ be an instance of Monotone One-In-Three Sat consisting of N variables $\mathcal{X} = \{x_1, \ldots, x_N\}$ and M clauses $C_1, \ldots, C_M$, with $C_i = (x_i^1 \vee x_i^2 \vee x_i^3)$, for $1 \leq i \leq M$.[1] We propose a gadget-based reduction that builds upon the following building blocks. Note that placing a building block at some y-coordinate means positioning the bottom-most site or port at that y-coordinate. To simplify the reduction, we create an instance that is not in general position. However, by moving the sites slightly, one can ensure general position without affecting the size or correctness of the reduction.

Blocker Gadget

The main purpose of the *blocker* gadget is, as the name suggests, to block leaders from crossing certain parts in the instance and thus ultimately splitting the generated instance into multiple segments. The blocker gadget consists of two smaller structures, each blocking one of the two sides of the boundary $\mathcal{B}$. These structures are composed of three sites and three ports, as shown in Figure 4.7. Furthermore, we have the grouping constraint $\{s_1, s_2, s_3\}$, and ordering constraints $s_1 \prec s_2$ and $s_2 \prec s_3$. Due to the grouping constraint, the labels of all sites will be on the same side of $\mathcal{B}$. However, due to the ordering constraints and the position of the sites, only one of the two sides is possible. In Figure 4.7, this would be the right side (even if ports would be on the left side), as indicated by the leaders. If we try to label the sites on the other side (assuming now that there were ports), we either end up with a non-planar labeling or have to violate one of the ordering constraints. By mirroring the layout of Figure 4.7, we can create a similar structure that enforces us to label the sites on the left side of the boundary. A blocker, positioned at some coordinate y, consists now of two such structures, S_A and

[1] We will use N and M to denote the number of variables and clauses, respectively, and n and m to denote the number of created sites and ports, respectively.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

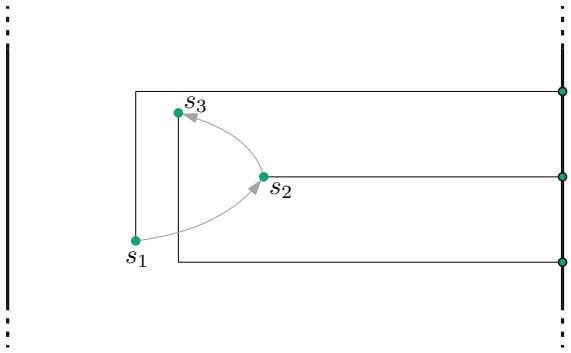


Figure 4.7: Structure that we use to make the blocker gadget. We also add the grouping constraint $\{s_1, s_2, s_3\}$ and the ordering constraints $s_1 \prec s_2$ and $s_2 \prec s_3$. The ordering constraints are visualized with gray arrows and the labeling with the leaders.

S_B, facing in opposite directions, one placed at $y + 1$, the other at $y + 4$. While the structures enforce the side they are labeled on, we also want to ensure that the labels have no room to move around on that side. Therefore, we place two sites, s_a and s_b, at a distance of 1 above and below the two structures, respectively, as shown in Figure 4.8. By adding the respective ports and two grouping constraints, one containing s_a and the sites of S_A, the other s_b and the sites of S_B, we ensure that the labeling for the blocker is completely dictated. Observe that any site above or below the blocker must be labeled above or below the blocker, respectively, i.e., there is no room for a leader to run through the blocker. Finally, we want to analyze the properties of the blocker.

Lemma 4.3. *A blocker B consists of eight sites, eight ports, four grouping, and four ordering constraints. Excluding the labels, it has a height of seven.*

Proof. For the number of sites and ports, we refer to the construction shown in Figure 4.8. Each blocker consists of two structures S_A and S_B, each having one grouping and two ordering constraints to ensure that the involved sites are labeled at the respective side of the boundary. The sites of S_A and S_B are also in a grouping constraint with s_a and s_b, respectively. Furthermore, each structure has a height of two, excluding the labels, as we need to maintain a distance of one between two consecutive ports to accommodate the labels.[2] We place the structures a distance of one apart, yielding a total height of five. We put the sites s_a and s_b, which bound the blocker from above and below, also at a distance of one above and below the two structures. Hence, the total height, excluding the labels, equals seven. □

In the upcoming sections, we will indicate a blocker B with a gray box that spans the whole width.

[2]Recall that we assume that the labels are open rectangles, i.e., they can touch at their borders.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

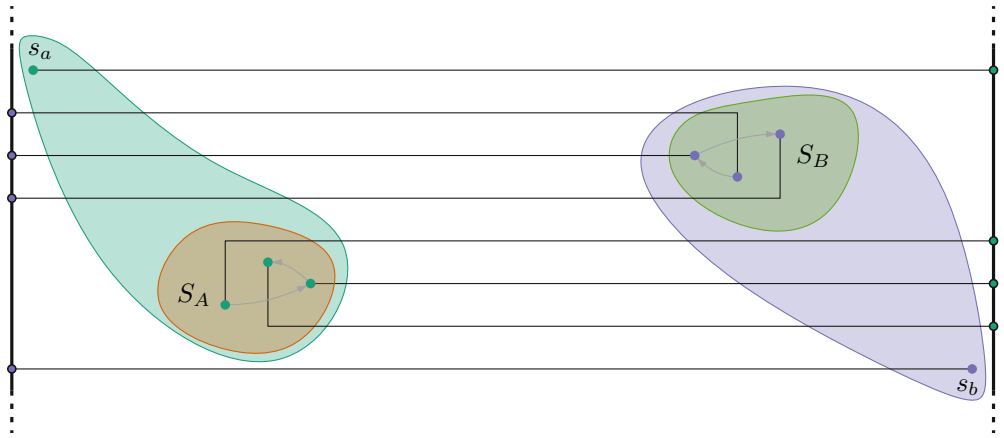


Figure 4.8: A blocker gadget B. We indicate the only possible labeling with the leaders.

Variable Block

The next ingredient of our reduction is the *variable* block. Its purpose is to select a truth assignment for each variable $x_i \in \mathcal{X}$, $1 \leq i \leq N$. We will determine the truth assignment of a variable by the side of $\mathcal{B}$ they are labeled at. To do that, we create in the variable block for each variable x_i a site s_i and two ports p_i^1 and p_i^0. We place the ports for s_i at the same y-coordinate as s_i, i.e., we have $y(s_i) = y(p_i^1) = y(p_i^0)$. The port p_i^1 is placed on the right side of the boundary. The presence of the leader $\lambda_i = (s_i, p_i^1)$ will indicate that x_i should be true. Similarly, p_i^0 is placed on the left side, and the presence of $\lambda_i' = (s_i, p_i^0)$ will indicate that x_i is false. We create this setup for all variables $x_i \in \mathcal{X}$ and stack them on top of each other, but leave a vertical space of one between two consecutive sites and ports to ensure that no two labels can overlap. Observe that there are no constraints on the sites representing the variables. Since we spread them at a sufficient distance, each site can be labeled either on the left or the right side, independent of the labeling of the other sites. Hence, for each variable assignment over $\mathcal{X}$, there is a corresponding labeling over $\{s_1, \ldots, s_N\}$. For the variable block, we furthermore observe the following.

Lemma 4.4. *The variable block consists of N sites, $2N$ ports, and no constraints. Excluding the labels, it has a height of $(N-1)$.*

Proof. Since we create for each of the N variables in $\mathcal{X}$ one site and two ports but no constraints, the first part of Lemma 4.4 readily follows. For the height of the variable block, we observe that between two consecutive sites, we leave a space of 1 to ensure that no two labels can overlap. Since we place the ports at the same vertical position as their respective site, we arrive at a height of $(N-1)$ for $N = |\mathcal{X}|$. □

Clause Gadget

The last gadget we will use in the reduction is the *clause* gadget. For a clause $C_i = (x_i^1 \wedge x_i^2 \wedge x_i^3)$, $1 \leq i \leq M$, it consists of three sites, c_i^1 to c_i^3, and three ports as depicted in

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

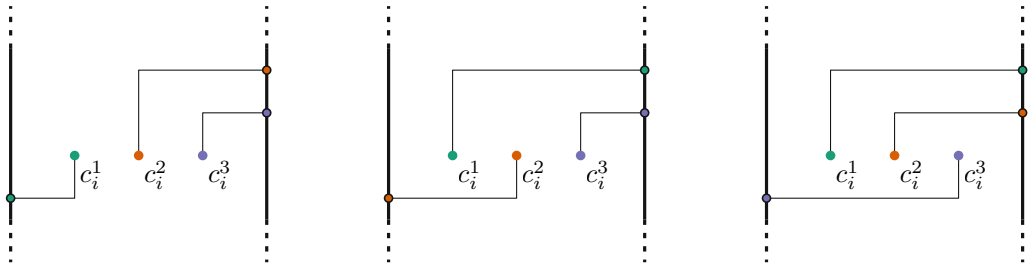


Figure 4.9: Clause gadget for the clause $C_i = (x_i^1 \wedge x_i^2 \wedge x_i^3)$ with its different labelings.

Figure 4.9. The purpose of the clause gadget is to check whether a given truth assignment on the variables satisfies the clause. Recall that a clause in Monotone One-In-Three Sat is satisfied if and only if exactly one of the tree literals evaluates to true, i.e., if exactly one variable present in C_i is set to true and the other two to false. To mimic this behavior in our setting, we place two ports on the right boundary and only a single port on the left boundary, such that any of the three sites can be labeled on the left side while ensuring that we can label the remaining two sites on the right side. Hence, in any feasible labeling, one label must be on the left side, and the corresponding site should then represent the variable that satisfies the clause. Observe that while in the variable block, the right side of the boundary represents true, here in the clause gadget it is the left side. The need for this asymmetry will become clear in Section 4.2.2 when we combine all building blocks. As for the other two gadgets, we end our presentation by arguing its properties.

Lemma 4.5. *The clause gadget for a clause C_i, $1 \leq i \leq M$, consists of three sites, three ports, and no constraints. Excluding the labels, it has a height of three.*

Proof. The number of sites and ports follows from Figure 4.9. Furthermore, observe that in the above description of the gadget, we do not have any constraints. Let the y-coordinate of the sites of the clause gadget for the clause C_i be some y_{C_i}. To ensure that any site can reach the single port on the left side of the boundary, we place it at $y_{C_i} - 1$. For the two ports on the right side of the boundary, we must ensure that we can place two labels there (without overlapping) and that we can label any two sites at these two ports. Therefore, we place the first one at $y_{C_i} + 1$ and the second at $y_{C_i} + 2$. In total, this gives us a height of three, excluding the labels. □

4.2.2 Putting It Together: Our Complete Construction

After presenting and discussing the building blocks of our reduction, it is time to combine them. Recall that we assume that we have given an instance φ of Monotone One-In-Three Sat consisting of N variables and M clauses, which we want to reduce to Existence of 2-Sided Constrained Boundary Labeling. Consider Figure 4.10 for

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

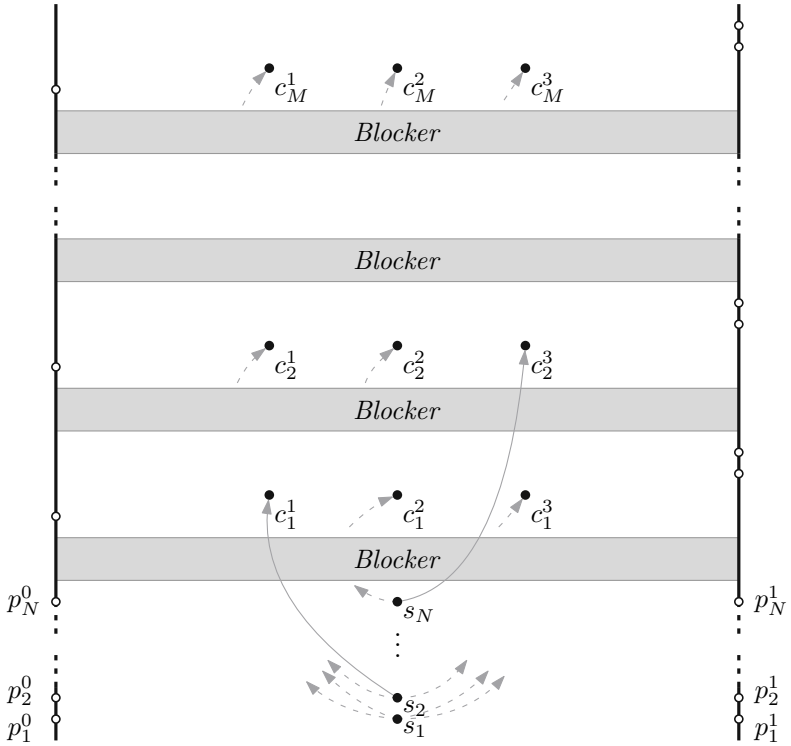


Figure 4.10: The instance $\mathcal{I}(\varphi)$ created by our reduction. The gray arrows indicate some of the ordering constraints in $\overline{\prec}$.

a visualization of the overall construction. The construction starts at the bottom with the variable block. Then, we add a blocker, followed by one clause gadget for each clause, created in arbitrary order, each separated by a blocker. We end the construction with the gadget for the last clause. Note that we must ensure that the vertical distance between any two ports of different gadgets is at least 1 so that no two labels placed in different gadgets can overlap. Having placed all the sites and ports, we still need to enrich the existing (ordering) constraints from the gadgets with additional ones. To do that, let $\varphi(x)$ denote all clauses that contain the variable x. We now add the following set of $3M$-many ordering constraints.

$$\overline{\prec} := \{s_i \prec c_j^l \mid x_i = x_j^l, \text{ where } l \in \{1,2,3\},\ x_j^l \in C_j,\ C_j \in \varphi(x_i),\ x_i \in \mathcal{X}\}$$

These ordering constraints, which are also partially depicted in Figure 4.10, can be summarized as enforcing for each variable $x \in \mathcal{X}$ that the label for the site it represents in the variable block must be before the labels of the sites that represent the occurrences of x in clauses in the respective clause gadgets. Our construction is now complete, and we denote with $\mathcal{I}(\varphi)$ the resulting instance of Existence of 2-Sided Constrained Boundary Labeling.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Theorem 4.3. Existence of 2-Sided Constrained Boundary Labeling *is NP-complete.*

Proof. We argue NP-containment and NP-hardness separately.

NP-containment. Given an instance $\mathcal{I}$ of Existence of 2-Sided Constrained Boundary Labeling, we can describe a feasible witness labeling $\mathcal{L}$ that respects the constraints as a function from the domain of the sites into the co-domain of the ports. Hence, it uses $\mathcal{O}(nm)$ space. To check whether $\mathcal{L}$ is planar, we evaluate all $\mathcal{O}(n^2)$ leader combinations. For the grouping constraints, we check for each of the k groups whether the corresponding $\mathcal{O}(n)$ sites are labeled on the same side of the boundary and no other label is between them. Finally, we can check the r ordering constraints in $\mathcal{O}(r) = \mathcal{O}(n^2)$ time. Therefore, we can check whether $\mathcal{L}$ is feasible in $\mathcal{O}(n^2 + kn)$ time. Since these are all polynomials in the input size, Existence of 2-Sided Constrained Boundary Labeling is in NP.

NP-hardness. We show NP-hardness using the above-introduced reduction from Monotone One-In-Three Sat to Existence of 2-Sided Constrained Boundary Labeling. To show the correctness of the reduction, we will use Observation 4.4. Let φ be an instance of Monotone One-In-Three Sat and $\mathcal{I}(\varphi)$ the corresponding instance of Existence of 2-Sided Constrained Boundary Labeling. Recall that we assume that φ consists of M clauses formed over N variables $\mathcal{X}$. Assuming that each clause gadget is "responsible" for the blocker immediately below it, we can see that we create in total M blocker gadgets. Using Lemmas 4.3 to 4.5, we derive that $\mathcal{I}(\varphi)$ consists of $8M + N + 3M = 11M + N$ sites, $8M + 2N + 3M = 11M + 2N$ ports, $4M$ grouping, and $7M$ ordering constraints. Observe that we have $4M$ ordering constraints in the blockers and $3M$ ordering constraints in $\overline{\prec}$. The height of $\mathcal{I}(\varphi)$, excluding labels, is $2M + N - 1 + 3M + 2M = 7M + N - 1$, where the last $2M$ follow from the vertical offset we have to maintain before and after each blocker so that they do not interfere with the other building blocks. $\mathcal{I}(\varphi)$ has a constant width.[3] Therefore, $\mathcal{I}(\varphi)$ has polynomial size and can be created in polynomial time with respect to the size of φ.

It remains to show the correctness of our reduction.

($\Rightarrow$) Assume that φ is a positive instance of Monotone One-In-Three Sat. Hence, there exists a truth assignment $\Gamma : \mathcal{X} \to \{0, 1\}$ such that for each clause $C_i \in C$, $1 \leq i \leq M$, we can find a $j \in \{1, 2, 3\}$ so that $\Gamma(x_i^j) = 1$ and $\Gamma(x_i^{j'}) = 0$, for $j' \in \{1, 2, 3\} \setminus \{j\}$, i.e., exactly one literal of each clause evaluates to true under Γ. We replicate this assignment in a labeling $\mathcal{L}$ of $\mathcal{I}(\varphi)$ by labeling s_i, $1 \leq i \leq N$, at p_i^1 if

[3] Observe that the actual x-coordinate of the sites is irrelevant in most gadgets. Only for the blocker gadgets, we must place their structures in such a way that they block any leader from running through them. Therefore, we can horizontally scale the instance arbitrarily. Thus, we assume, without loss of generality, that the width of the instance is constant.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$\Gamma(x_i) = 1$, otherwise at p_i^0. We label the sites in the clause gadget at the left boundary if the corresponding variable they represent satisfies the clause, i.e., is true, and otherwise at the right side of the boundary. For the sites that make up the blocker gadgets, we label them according to Figure 4.7. What is left to do is argue that $\mathcal{L}$ is feasible. For the sites in the blocker gadgets, this is true by construction. As for any variable $x \in \mathcal{X}$, we have either $\Gamma(x) = 1$ or $\Gamma(x) = 0$, but never both, the label positions for the sites representing the variables are well-defined. Since we placed the ports and sites with sufficient space from each other, mimicking Γ in $\mathcal{L}$ as described above will always result in a feasible labeling. For the sites in the clause gadgets, as Γ ensures that exactly one literal evaluates to true, we know that one site will be labeled at the left boundary and the other two at the right boundary. This is exactly the distribution of the three ports we have chosen when creating the clause gadgets (see Figure 4.9). Furthermore, we have ensured that no matter which two sites we label on the right side, there is a way to label them. Therefore, $\mathcal{L}$ is planar. To show that $\mathcal{L}$ respects also the constraints, we first note that we respect the constraints in the blocker gadget by construction. Therefore, we only have to consider the ordering constraints from $\overline{\prec}$. Let $s_i \prec c_j^l$, $1 \leq i \leq N$, $1 \leq j \leq M$, $l \in \{1, 2, 3\}$, be an ordering constraint from $\overline{\prec}$. Since s_i represents a variable and c_j^l its occurrence in a clause, we know that s_i will be labeled below c_j^l in $\mathcal{L}$. Therefore, to respect $s_i \prec c_j^l$, we must show that s_i and c_j^l are labeled on different sides, i.e., that we deactivate that constraint. There are two cases: $\Gamma(x_i) = 0$ or $\Gamma(x_i) = 1$. In the former case, we label s_i on the left side and, as the clause C_i is then not satisfied by the variable x_i, c_j^l is labeled on the right side. In the latter case, i.e., if $\Gamma(x_i) = 1$, we label s_i on the right side and, as it satisfies the clause, c_j^l is (the only site in this clause gadget that is) labeled on the left side. As we label them in both cases on different sides of the boundary, we deactivate the corresponding constraint. Since we selected $s_i \prec c_j^l$ arbitrarily, we know that we deactivate all constraints in $\overline{\prec}$. Hence, $\mathcal{L}$ is feasible, thus is $\mathcal{I}(\varphi)$ a positive instance of Existence of 2-Sided Constrained Boundary Labeling.

($\Leftarrow$) Let $\mathcal{I}(\varphi)$ be a positive instance of Existence of 2-Sided Constrained Boundary Labeling. Therefore, there exists a feasible witness labeling $\mathcal{L}$. Based on $\mathcal{L}$, we create a truth assignment $\Gamma : \mathcal{X} \to \{0, 1\}$ over $\mathcal{X}$ with $\Gamma(x_i) = 1$ if $\mathcal{L}$ labels s_i at p_i^1, $1 \leq i \leq N$. Otherwise, i.e., if $\mathcal{L}$ labels s_i at p_i^0, we set $\Gamma(x_i) = 0$. Due to the structure of the variable block, we can assume that we label each s_i at one of those two ports, and thus is Γ well-defined. All that remains is to show that in every clause C_i exactly one literal evaluates to true. We remind the reader that every literal in φ is an un-negated variable. $\mathcal{L}$ respects all the constraints of $\mathcal{I}(\varphi)$, in particular the ordering constraints $\overline{\prec}$. However, observe that due to the blocker gadgets, especially the one between the variable block and the first clause gadget, in any feasible labeling, and therefore also in $\mathcal{L}$, all sites in the variable block are labeled below those in the clause gadgets. Therefore, to respect the constraints in $\overline{\prec}$, $\mathcal{L}$ must deactivate all of them. Hence, for each site s_i in the variable block that we label at the right side, i.e., at p_i^1, we must label all sites c_j^l, $1 \leq j \leq M$, $l \in \{1, 2, 3\}$, with $s_i \prec c_j^l \in \overline{\prec}$ at the left side. A symmetric argument holds if we label s_i on the left side. However, since for every clause gadget representing

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

some clause C_j, $1 \leq j \leq M$, we create three ordering constraints in $\overline{\prec}$ and there is only one port on the left side and two ports on the right side, we know that $\mathcal{L}$ can only be a feasible labeling if exactly one of the three sites that make up the clause gadget is labeled on the left side, i.e., considered true. Consequently, Γ satisfies exactly one literal of each clause in φ, i.e., φ is a positive instance of Monotone One-In-Three Sat. □

Corollary 4.3. 2-Sided Constrained Boundary Labeling *is NP-hard.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Experimental Evaluation

The previous chapters shed light on the b-Sided Constrained Boundary Labeling problem from a theoretical point of view. Given that the motivation for this problem arises from real-world applications, we also want to consider the practical aspects of this problem. Therefore, we implemented our DP-Algorithm. In an experiment, we will analyze its performance and assess the produced labelings regarding leader length and visual appeal. We also compare the created labelings to those from an alternative algorithm that does not consider grouping and ordering constraints.

Section 5.1 explains the implementation in greater detail and describes how we can interact with the solvers and visualize the labelings. We will discuss the setup of the experiment in Section 5.2 and its results in Section 5.3. Throughout this chapter, we will state distances and coordinates in *pixels*.

5.1 Implementation Details

Our implementation consists of two parts: two *C++17* solvers, which compute the labeling, and a web application to visualize them.

5.1.1 Solvers

Our first solver, *Naïve ILP*, is based on the Integer Linear Program (ILP) formulation by Barth et al. [BGNN19]. It does not consider our grouping and ordering constraints and should provide a leader-length-minimal labeling to compare against. The second solver, *Constraint DP*, implements the dynamic programming algorithm we presented in Chapter 4. Both solvers expect the instances as *JavaScript Object Notation* (*JSON*) files. Each JSON file describes a single instance and contains solver-relevant information on the position of the sites and ports, the constraints, and the label height. Furthermore, it

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

contains information purely relevant for visualizing the instance, such as the position and the size of the boundary, the contour of the illustration, or the width of the labels.

Naïve ILP

Our solver Naïve ILP uses an ILP formulation to find a leader-length-minimal feasible labeling. ILPs find application in many areas, for example, scheduling [FL05; JSV98] or facility location [CNW83]. An ILP formulation consists of variables, an optimization function, and constraints. The main idea is to encode our problem using linear (in)equalities over the variables, i.e., there is no multiplication among the variables, such that each assignment of values to the variables that satisfies all (in)equalities corresponds to a feasible solution. We can then use the optimization function to quantify the quality of our solutions and select the best one. Wolsey [Wol20] gives a more detailed introduction to integer programming.

ILP Formulation. We use the ILP formulation from Barth et al. [BGNN19] as a reference and extend it by constraints that prevent overlapping labels and site-leader crossings. More concretely, for an instance $\mathcal{I}$ of 1-Sided Constrained Boundary Labeling, we create nm-many binary variables $x_{s,p} \in \{0,1\}$, for $s \in \mathcal{S}$ and $p \in \mathcal{P}$. If we label s at port p, we set $x_{s,p} = 1$. Otherwise, we have $x_{s,p} = 0$. Let $f : (s,p) \to \mathbb{R}_0^+$ express the length of the po-leader $\lambda = (s,p)$. We minimize the function

$$\sum_{\substack{s\in\mathcal{S},\\ p\in\mathcal{P}}} f(s,p)x_{s,p}$$

subject to the following constraints.

- Each site s must be labeled at exactly one port.

$$\sum_{p\in\mathcal{P}} x_{s,p} = 1, \quad \forall s \in \mathcal{S}$$

- Each port p can be used by at most one site.

$$\sum_{s\in\mathcal{S}} x_{s,p} \leq 1, \quad \forall p \in \mathcal{P}$$

- No two labels can overlap, i.e., if two ports are too close, then at least one must not have a label.

$$\sum_{s\in\mathcal{S}} x_{s,p} + x_{s,p'} \leq 1, \quad \forall p,p' \in \mathcal{P},\ p \neq p',\ |y(p) - y(p')| < h$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- No two leaders can cross, i.e., if two leaders would cross, the labeling must not contain both.

$$x_{s,p} + x_{s',p'} \leq 1, \quad \forall s, s' \in \mathcal{S},\ \forall p, p' \in \mathcal{P},\ s \neq s',\ p \neq p',\ \text{where leaders } \lambda = (s,p) \text{ and } \lambda' = (s',p') \text{ cross, i.e., } x(s) < x(s') \text{ and } \lambda' \text{ separates } s' \text{ and } p'$$

- A site s cannot be labeled at a port p with the same y-coordinate as another site s' that is right of s, as this would mean that $\lambda = (s,p)$ crosses s'.

$$x_{s,p} = 0, \quad \forall s \in \mathcal{S},\ \forall p \in \mathcal{P},\ \text{if there exists an } s' \in \mathcal{S} \text{ with } s \neq s',\ x(s) < x(s'),\ |y(s') - y(p)| < \varepsilon,\ \text{for some sufficiently small } \varepsilon > 0$$

Recall that we use this solver to provide a leader-length-minimal feasible labeling as a reference. Therefore, we do not incorporate inequalities for our semantic constraints.

As a solver, we use the *Gurobi Optimizer* (Version 10.0.0) [Gur23a]. In addition to the labeling, if it exists, we also store the sum of the leader lengths, the time spent defining the model, and the time it takes Gurobi to solve it.

Command-Line Interface. Naïve ILP can be started from the command line using `.\Naive-ILP PATH_TO_INSTANCE.json`. It furthermore supports the following optional arguments.

`-n, --name`: Specifies under which name the labeling should be displayed in the visualization. Default is the same name as the instance.

`-o, --output`: Determines the path to the output file. If omitted, we store it next to the instance file.

`-v, --verbose`: Enables verbose mode, where we output more detailed information.

`--ilp_log`: Specifies the path where the Gurobi Log-File should be created; see the documentation [Gur23c] for further information. If omitted, no log will be stored.

`--ilp_model`: Specifies the path where the created ILP model should be saved; see the documentation [Gur23b] for further information. If omitted, the model will not be saved.

Constraint DP

The solver Constraint DP implements our proposed algorithm to find a leader-length-minimal feasible labeling that respects the constraints or to report that no such labeling exists. As for Naïve ILP, we also keep track of the sum of the leader lengths and the running time of various parts of the algorithm. The algorithm follows the description of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Chapter 4. We use the range tree implementation by Weihs [Wei20] that was already used in the literature [WDM18] but fix the compilation warnings that we observed. In the following, we discuss noteworthy deviations from the description from Chapter 4.

Implementing the PQ-A-Graph. Our PQ-A-Graph implementation builds on top of an existing *PC-Tree* implementation by Fink et al. [FPR21]. Shih and Hsu introduced PC-Trees to check whether a graph is planar [SH99]. A formal description of PC-Trees is given by Hsu and McConnell [HM03]. PC-Trees are trees that contain two types of nodes: *P-nodes*, which allow the same operations on their children as their related nodes in PQ-Trees, and *C-nodes*, where we have to maintain the cyclic order of their children or can inverse it [SH99]. Note that this operation is more general than the inversion allowed by Q-nodes [Hsu01]. In contrast to PQ-Trees, PC-Trees can also be unrooted. This allowed Hsu and McConnell [HM03] to show that we can use PC-Trees to represent PQ-Trees by introducing a dummy element that does not occur in any constraint. After computing the respective PC-Tree, we can find the leaf for this dummy element and interpret it as the root of the tree, whose single child is then the root of the corresponding PQ-Tree if we re-interpret the C-nodes as Q-nodes. This result allows us to use (rooted) PC-Trees instead of PQ-Trees, for which there are (in practice) more efficient implementations [FPR21].

Practical Considerations. Our initial runs revealed that a naïve implementation of the DP-Algorithm has high running times even for small instances. We identified two major causes for the high running time that we want to describe in the following.

First, it turned out that it is beneficial to explicitly pre-compute the least common ancestor of all possible leaf pairs and look up this information in the DP-Algorithm. The following approach was sufficient for our needs. We start at the root of $\mathcal{T}$ and proceed downwards in the tree. For each internal node t, we consider each pair t_i and t_j, $i \neq j$, of its children and store t as the least common ancestor of each pair of leaves $l_i \in \text{leaves}(T_i)$, and $l_j \in \text{leaves}(T_j)$. Since we consider each pair of leaves only once, this takes $\mathcal{O}(n^2)$ time.

Second, we could observe that especially instances containing segments on the boundary that have many ports but few sites suffered from high running times. One reason for this is that although placing leaders at the different ports in these segments affects the leader length of the resulting labeling, the resulting sub-instances look similar from a geometric perspective. Consider, for example, Figure 5.1. Regardless of whether we label s at p_x or p_y, the resulting sub-instances I_2^x and I_2^y contain the same sites, have the same site s' as its leftmost site, and (almost) the same ports. Therefore, it is not necessary to evaluate again for every port in I_2^y in its entirety whether we can place the label for s' there if we have performed these checks already in I_2^x. More concretely, assume that we observe that the bounding box of the sites in an instance $I = (s_1, p_1, s_2, p_2)$ is identical to the one in the instance $I' = (s_1, p_1', s_2, p_2')$. Then, due to our general position assumption, we know that the same sites must be inside that bounding box. Furthermore,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

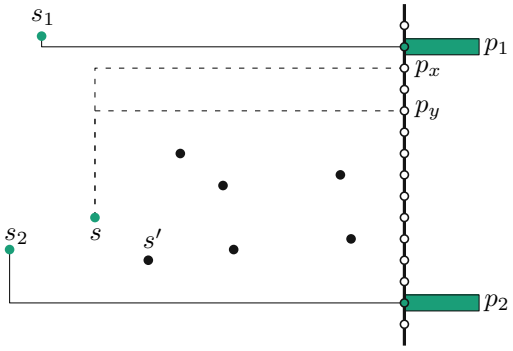


Figure 5.1: No matter whether we label s at p_x or p_y, a leader $\lambda = (s', p)$ will in both resulting sub-instances I_2, bounded by s and s_2, either respect the constraints or do not respect them.

since both instances are described by s_1 and s_2, everything that affects the outcome of RespectsConstraints$(I, \mathcal{T}, \lambda = (\text{s, p}))$ is identical, given that s is the leftmost site in I (and I') and we have $p \in (\mathcal{P}(I_1) \cap \mathcal{P}(I_2))$. This gives rise to using memoization to run RespectsConstraints$(I, \mathcal{T}, \lambda = (\text{s, p}))$ only when we encounter a new geometric structure of the (sites of an) instance. Note that we still have to ensure for all generated sub-instances that there are sufficiently many ports in the instance. Furthermore, we want to mention that Niedermann et al. [NNR17] also faced the problem of similarly structured sub-instances, which they solved by bundling them together.

Command-Line Interface. As for Naïve ILP, we can start Constraint DP from the command line using `.\Constraint-DP PATH_TO_INSTANCE.json`. The program supports the optional arguments `-n` or `--name`, `-o` or `--output`, and `-v` or `--verbose`, with the same semantics as Naïve ILP. In addition, it also supports the following optional argument.

`-i, --inverse_ordering`: If present, an ordering constraint $s \prec s'$ will be interpreted as $s' \prec s$, i.e., inversed. This is convenient for the visualization to maintain the semantics of ordering constraints visually, as the y-axis points downwards in an SVG [DDG+11, Chapter 7].

5.1.2 Visualization

We also provide a complementary visualization of the computed labelings. The core functionality of the visualization is written in *D3.js* (Version 7.8.5) [BOH11] and embedded in an *Angular* (Version 16.1.5) [Ang23] web application. As for the solvers, we use JSON files to store the computed labelings. A single file can store several labelings for the same instance, for example, computed with different (settings of) solvers. Each stored labeling not only describes the leaders, but contains further information, such as the total leader length of the labeling, used solver, and running times.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

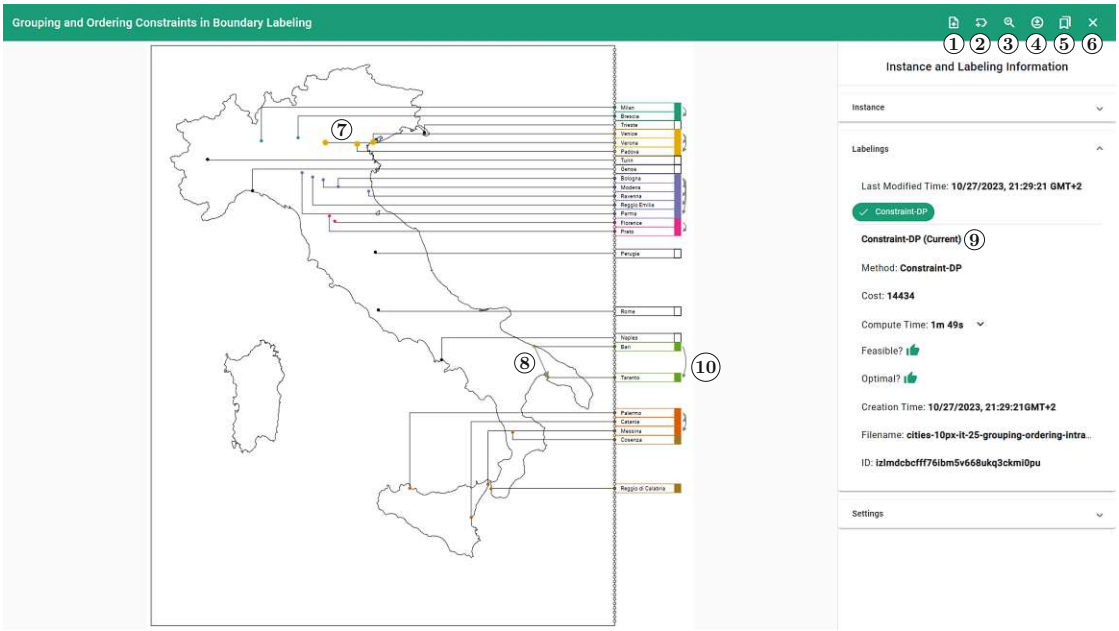


Figure 5.2: A screenshot of our visualization. It shows an instance of the cities-10px dataset with intra-ordering constraints and the labeling created by Constraint DP.

User Interface

We provide a screenshot of the user interface in Figure 5.2. The core of it is the central canvas area where the instance and the labeling, i.e., the figure, are shown. We provide several options to interact with the figure and to retrieve further information about it.

Interaction Possibilities. There are several buttons located in the top bar. We can use them to load an instance file (①) or a labeling file for the current instance (②). Furthermore, we can zoom in or out of the figure (③), download the currently shown figure as a `.png` or `.svg` file (④), load sample instances and their corresponding labelings (⑤), or open and close the right sidebar (⑥).

While the top bar is to determine what should be shown on the canvas area, the aforementioned sidebar provides us with further information about the figure we see. It is divided into three sections, providing information about the *Instance*, the *Labelings*, or controlling in the *Settings* their appearance.

In the *Instance* section, we list information regarding the currently loaded instance. Hovering over a site or a port visualizes the respective `id`. We can copy this `id` to the clipboard by clicking on the site or port. To visualize a grouping or ordering constraint, we can hover over it. For a grouping constraint, we enlarge the involved sites (⑦). Ordering constraints are visualized with an arrow (⑧). With a plus-button next to the grouping constraints, we can interactively define a new grouping constraint by clicking

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

on the sites that should be part of the group. Once we have included all sites, we can copy the respective JSON snippet into the clipboard by clicking a checkmark.

In the *Labelings* section, we can retrieve the information about the loaded labelings. There is a gray chip for each labeling from the JSON file. If the labeling is feasible, we can click on the chip to load the labeling into the illustration, i.e., complete the figure. We mark the currently loaded labeling with *(Current)* next to the information (⑨). For example, in Figure 5.2, we have loaded a labeling created with the solver Constraint DP.

In the *Settings* section, we can decide whether we want to show the ports. Furthermore, we can enable a different way of visualizing the constraints. On the one hand, we can color each site (and its label) to indicate the grouping constraints by color or leave them black (white) to indicate that they are not part of a grouping constraint. On the other hand, we can embed the arcs for the ordering constraints next to the respective labels (⑩). Note that the former visualization is only correct if there are at least $|\mathscr{G}|$ different colors and we have non-overlapping groups, as we color each site in a single color. We provide a default set of sixteen colors but the user can also set an individual color palette. The latter visualization is only applicable if we enrich our illustration by a labeling.

5.2 Setup of the Experiments

Our experimental evaluation can be divided into four types of experiments, each with its own dataset, to analyze different properties of the problem and our algorithm. In the following, we describe first the datasets and the questions we want to answer with them before we state the used hardware. Note that we already stated the used software when describing our implementation in Section 5.1.

5.2.1 Datasets

Unless stated otherwise, we created the datasets using auxiliary Python scripts (Python Version 3.8.10). We ensured general position for all datasets by moving sites in steps of one pixel in x or y direction, where necessary. Furthermore, we will, in the following, differentiate between *intra-group* and *inter-group* ordering constraints. The former are ordering constraints among sites of the same group, thus creating partial orders within the groups. The latter are ordering constraints among sites of different groups, thus enforcing that an entire group must be labeled above another group. Unless stated otherwise, we place the boundary and the figure with an offset of ten pixels in each direction around the object they enclose. Furthermore, we ensure that the size of the figure is large enough to accommodate the labels that have a height of twenty pixels and a width of 150 pixels. The height is selected such that a single-line text with a font size of twelve points can be accommodated inside the label. The width was selected to accommodate a typical single-word text. However, note that the width of the label is only set for visualization purposes and has no influence on the feasibility of an instance or the running time to compute a labeling. In each instance, we place the ports on the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

right side of the boundary. They are spaced out evenly along the side of the boundary, maintaining a minimum distance between each other that varies between the datasets. If the height of the boundary is not large enough to accommodate all ports, we enlarge it accordingly. We round the computed coordinates and sizes, i.e., widths and heights, down to integers, where necessary.

Cities-*

The aim of the *cities* datasets is to analyze the impact of the number of sites on the running time of our algorithm and the quality of the resulting labelings. It consists of instances containing the $n \in \{10, 15, 20, 25, 30, 35, 40, 45\}$ largest cities from Austria, Germany, and Italy, respectively, obtained from `simplemaps.com` [Sim23]. We enrich the instances with contours of the countries extracted from *Wikimedia Commons* images [Nor23a; Nor23b; Nor23c]. Each combination of country and number of cities results in four different instances. One instance is without any constraints and intended to be solved by Naïve ILP, acting as a baseline for reference. We group the sites in the other three instances according to the administrative regions of the respective country. In addition, one instance contains intra-group ordering constraints, where we order the cities according to their administrative status. The capital of an administrative region should, for example, be labeled above the other cities in this region. The fourth instance contains inter-group ordering constraints, where we order the groups according to their population computed from the cities in the instance. Depending on the dataset, we create a different number of ports. In *cities-2x*, each instance has $m = 2n$ many ports, and *cities-90* defines 90 ports per instance. For these two datasets, we maintain a distance of at least twenty pixels, the label height h, between two ports. For *cities-10px*, we perform differently. We take the initial height of the boundary and place a port every ten pixels. Note that in a labeling every second port cannot be used due to the height of the labels. If this leads to too few ports, we add more ports accordingly, thus increasing the height of the boundary. This approach led to 71 ports for the Austrian cities and 128 and 130 for the German and Italian cities, respectively. Each dataset contains 96 different instances.

Ports

The second dataset, *ports*, is a variation of the cities dataset(s) and should analyze the effect of the number of ports on the running time, feasibility, and quality of the labelings. To do that, we consider the $n = 25$ largest cities from the countries and with the constraints as described for the cities dataset(s). For each combination of country and constraints, we create different instances where we vary the number of ports m. We use $m \in \{\lceil xn \rceil \mid x \in \{1.0, 1.1, \ldots, 2.9, 3.0\}\}$ and enforce a distance of at least ten pixels between two ports. This results in 252 instances for this dataset.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Figure	n	m	k	r	Definition of Constraints
Fig. 8.30	11	44	3	0	Groups originating from colored regions or curly brackets enclosing labels.
Fig. 8.81	9	60	0	7	A nerve branching off another nerve is labeled below its "parent".
Fig. 9.23	11	53	3	0	Overlapping groups based on explicit curly brackets or colored regions.
Fig. 12.33	9	38	3	0	Grouping based on colored regions. Sites on the boundary of two regions are in both regions, i.e., groups overlap.
Fig. 12.59	19*	62	3	6	Grouping based on curly brackets, ordering based on Roman letters next to some labels.

* The original figure contains 31 sites labeled on the left and right sides of the illustration. However, we took only the sites labeled on the left side.

Table 5.1: Properties of the human anatomy dataset.

Random

The *random* dataset is based on artificial data and consists of 200 randomly generated instances. We consider $n \in \{15, 20, 25, 30, 35\}$ different uniformly placed sites on an 800×800 pixels plane.[1] Each instance contains $k \in \{0, 2, 8\}$ grouping and $r \in \{0, 5, 10\}$ ordering constraints. To generate the grouping constraints, we randomly choose k different sites acting as the *roots* of the groups. To simulate the spatial proximity of sites in the same group, we consider all sites within a 200-pixel radius around each root. Each such site has a 75% chance of being part of the group. The size of the resulting groups varies between one and eleven, with a mean of 3.82. Two groups of the same instance overlap on average in 1.05 sites. For the ordering constraints, we randomly select two sites. We create five instances for each combination of n, k, and r, with $k + r > 0$, to balance out random factors. In each instance, we place a port every ten pixels.

Human Anatomy

The fourth dataset, *human anatomy*, is a small dataset containing only five instances. It contains instances obtained from the Sobotta atlas of human anatomy [WP13] that Niedermann et al. used to evaluate the performance of their contour labeling algorithm [NNR17]. These instances are enriched with grouping and ordering constraints. The constraints are either explicitly expressed in the book, for example, by curly brackets, or deemed as reasonable by us, for example, due to colored regions in the figures or the branches of nerve fibers. Note that the book uses contour labeling for their figures. Therefore, we select the instances such that we maintain feasibility also under the boundary labeling

[1] The actual plane has a size of 1000×1000 pixels, but we ensure that the sites keep a distance of at least 100 pixels to the border.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

model. Similar to Niedermann et al., we place a port every ten pixels. We summarize the properties of the instances in Table 5.1. The motivation behind this dataset is to analyze our algorithm in real-world scenarios. We, therefore, will compute the labelings for this dataset on an off-the-shelf laptop.

5.2.2 Experimental Setup

The experiments with the cities-*, ports, and random datasets were executed on the compute cluster of the *Algorithms and Complexity* group at *TU Wien*. The compute cluster consists of 16 nodes, each having two *Intel® Xeon® E5-2640 v4, 2.40GHz 10-core* processors backed up by *160 gigabytes* of RAM [Alg23]. We set a hard memory limit of 96 gigabytes that we never reached. In addition, the time limit of twenty hours per instance was also never exceeded.

The instances of the human anatomy dataset were run on a laptop with an *Intel® Core™ i5-8265U, 1.60GHz 4-core* processor. The laptop runs *Windows 11 Pro 22H2* and has *16 gigabytes* of RAM. For compatibility reasons, we ran the experiments inside a *WSL2* environment of *Ubuntu 20.04.6 LTS* that has *7865 mibibytes* of RAM available.

We measure the running time of our solvers in wall-clock time. In the measurement, we exclude the reading (and parsing) of the instance and writing of the labeling but include any other preprocessing steps, such as creating the PQ-A-Graph.

5.3 Results

In the following, we present and discuss the results of our experiments. We analyze them in Sections 5.3.1 to 5.3.4 for each dataset and give in Section 5.3.5 a final verdict.

5.3.1 Cities-*

We start with the results for the instances from the cities datasets. Table 5.2 presents the fraction of feasible instances for each dataset grouped by country and type of constraints, where we indicate the presence of the latter with the checkmarks and crosses. Almost every second instance is infeasible. However, we can see that the infeasibility is not evenly distributed across the different groups of constraints and concentrated in the instances containing ordering constraints. A reason for this is the location and the size of the groups. One of the regions with the most inhabitants is, for example, in Italy *Latio*, which is located in the center of Italy. Due to the inter-ordering constraints, we would need to label sites in this group above most other sites, blocking most ports with their leaders. Many instances with grouping constraints are feasible, probably due to the spatial proximity of the sites within a group. One exception is Germany, where probably the group *North Rhine Westphalia*, which has many cities, turned the instances infeasible.

We proceed with analyzing the running time of the algorithm and the leader length of the labelings. Since most instances with ordering constraints are infeasible, we do not include

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Country	Constraints in the Instances			# Feasible / # Total Instances		
	Gr.*	Intra Or.†	Inter Or.‡	cities-2x	cities-90	cities-10px
Austria	✔	✘	✘	8 / 8	8 / 8	5 / 8
	✔	✔	✘	1 / 8	1 / 8	1 / 8
	✔	✘	✔	0 / 8	0 / 8	0 / 8
	✘	✘	✘	8 / 8	8 / 8	8 / 8
Germany	✔	✘	✘	3 / 8	8 / 8	4 / 8
	✔	✔	✘	0 / 8	0 / 8	1 / 8
	✔	✘	✔	0 / 8	0 / 8	0 / 8
	✘	✘	✘	8 / 8	8 / 8	8 / 8
Italy	✔	✘	✘	7 / 8	7 / 8	7 / 8
	✔	✔	✘	3 / 8	3 / 8	4 / 8
	✔	✘	✔	0 / 8	0 / 8	0 / 8
	✘	✘	✘	8 / 8	8 / 8	8 / 8
Austria		–		17 / 24	17 / 24	14 / 24
Germany		–		11 / 24	16 / 24	13 / 24
Italy		–		18 / 24	18 / 24	19 / 24
–	✔	✘	✘	18 / 24	23 / 24	16 / 24
–	✔	✔	✘	4 / 24	4 / 24	6 / 24
–	✔	✘	✔	0 / 24	0 / 24	0 / 24
–	✘	✘	✘	24 / 24	24 / 24	24 / 24
	Total:			**46 / 96**	**51 / 96**	**46 / 96**

* Grouping constraints
† Intra-group ordering constraints
‡ Inter-group ordering constraints

Table 5.2: Feasible instances in the cities datasets.

them in the upcoming evaluation. However, we compared, for the feasible instances with ordering constraints, the running time and leader length to the corresponding instance with only grouping constraints. Regarding the running time, we could not observe a clear pattern. The leader length was, on average, at most one percent higher across all three datasets. This could imply that these labelings differ in re-routing a few leaders only.

Leader Length

Figure 5.3 plots the relative increase of the leader length of the (feasible) instances with grouping constraints to the corresponding instances without constraints. We can see that for the Austrian and Italian instances, the relative increase is across all datasets less than 1.15. In particular, the increase for the Italian cities is negligible. Noteworthy is

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

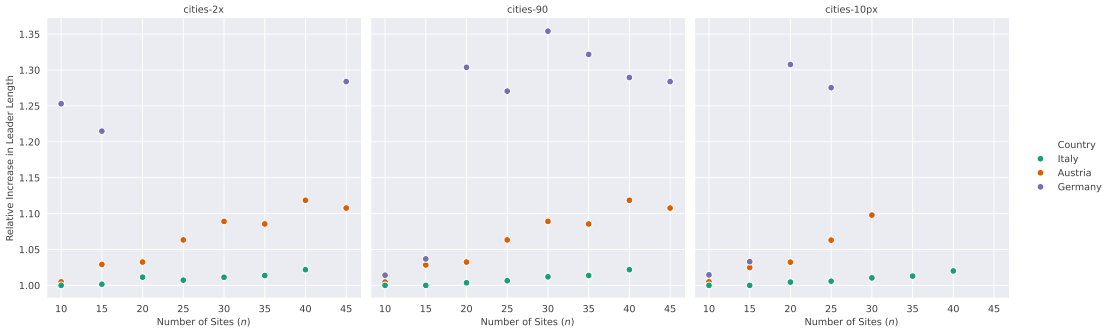


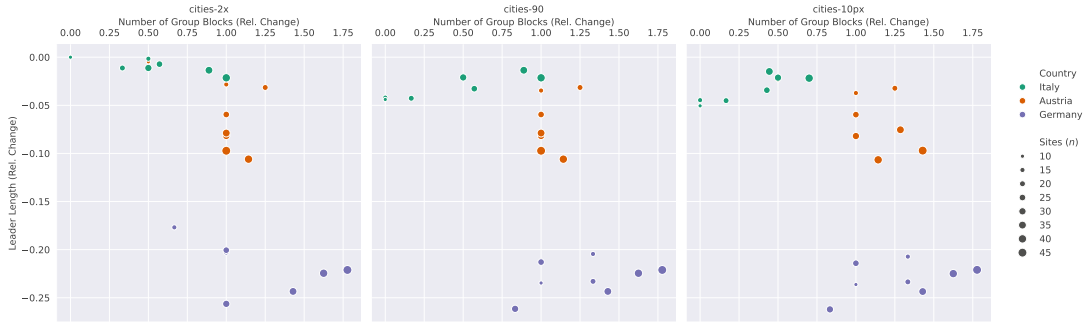
Figure 5.3: Relative increase of the leader lengths obtained with Constraint DP when compared to labelings from Naïve ILP for instances from the cities datasets.

also that we have in the beginning almost no increase, and only for larger instances it becomes significant for Austrian instances. One reason for this is that in denser instances we might need to label entire (large) groups at other places, thus increasing the leader length significantly. Note that the absence of points in Figure 5.3 means infeasibility of the corresponding instance. We can see that for the instances with German cities (see the violet observations in Figure 5.3) in cities-2x, only the first two instances and the last one were feasible. Although a bit counterintuitive at first, one possible explanation for this is that adding to this instance some sites means adding twice as many ports. Hence, once enough ports are there, the instance becomes feasible.

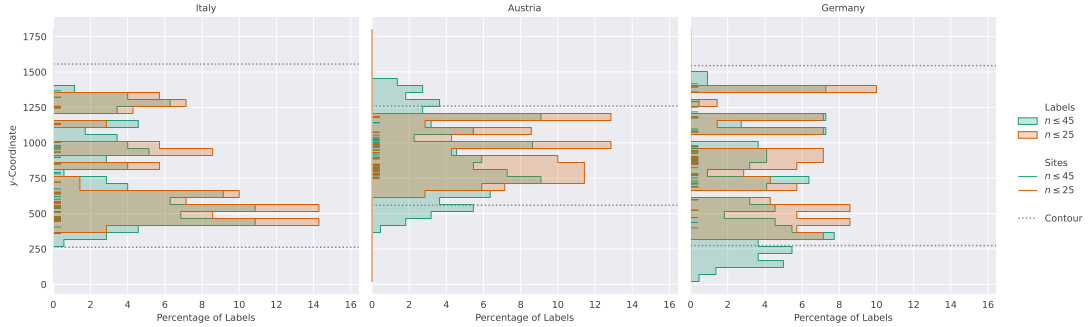
We were also interested in analyzing whether the spatial proximity of sites alone is sufficient to respect the grouping constraints, at least to some extend. To do that, we let $\mathcal{L}$ be the labeling reported by Constraint DP and $\mathcal{L}'$ the one by Naïve ILP. We compute, for an evaluation function g, the relative change as $\frac{g(\mathcal{L}')-g(\mathcal{L})}{g(\mathcal{L})}$, i.e., we take our algorithm as the baseline. In Figure 5.4a, we plot this relative change for the number of group blocks against the relative change in leader length. To compute the *number of group blocks*, we count for each group the number of maximal sets of consecutive labels it is composed of. Note that this value must be at least k, the number of groups, and splitting a group increases it by one. Figure 5.4a reveals that many instances lie, with respect to the leader length, in the first third of the plot. However, the observations are rather scattered concerning the group blocks, and we can observe that larger instances are further to the right. Hence, in the labeling created by Naïve ILP, the groups are sometimes considerably split up while the labelings have only five to ten percent shorter leaders. This could indicate that the price to pay, in terms of leader length, to respect the grouping constraints is small compared to how much more understandable such labelings are as opposed to ones that only optimize for leader length. However, the instances created from German cities do not follow the above-mentioned pattern, and it seems that the tradeoff of respecting the constraints is higher. To let the reader judge the visual quality of the created labelings, we present in Figures 5.7 to 5.9 some of the them.

To better understand this phenomenon, we want to analyze where the labels are placed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) The tradeoff of respecting the semantic constraints in the cities datasets.



(b) The distribution of the sites and ports used by labelings in the cities-90 dataset.

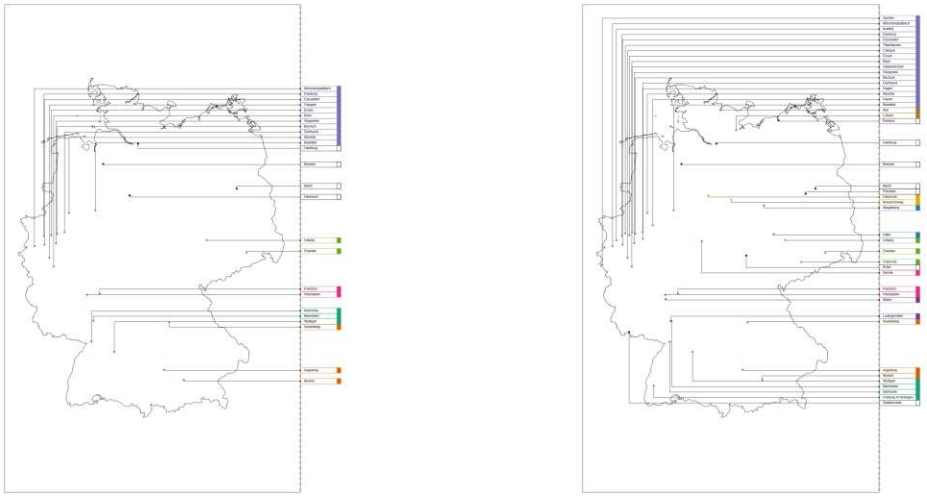
Figure 5.4: Impact of the semantic constraints on the placement of the labels and the resulting leader lengths in the cities datasets.

As cities-90 has the most feasible instances, we take this dataset and subdivide each instance into fifty equal-height rows between the first and the last port. Each row contains roughly the same number of ports, and we count for each row how many times we placed a label there. The resulting histogram is given in Figure 5.4b. Furthermore, we plot the y-coordinate of the sites and the extent of the contour. To see changes with increasing n, we visualize this once for all instances with $10 \leq n \leq 25$ and once for $10 \leq n \leq 45$. It can be observed that y-ranges with many sites are not necessarily more heavily used by the labels. For Germany, this is, for example, due to *North Rhine Westphalia*, as Figure 5.5a reveals. Finally, note that for small n, all labels are placed inside the contour, whereas this is not the case for large n. To see this for Germany, compare the labelings of Figure 5.5.

Running Time

We also want to shed light on the time required to compute the labelings. Figure 5.6a contains running time plots for all datasets. We want to remind the reader that the number of ports in cities-2x is not constant. Hence, in Figure 5.6a, varying the number of sites also influences the number of ports. For all other datasets, the number of ports is the same across all instances from the same country. Furthermore, recall that we do not

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



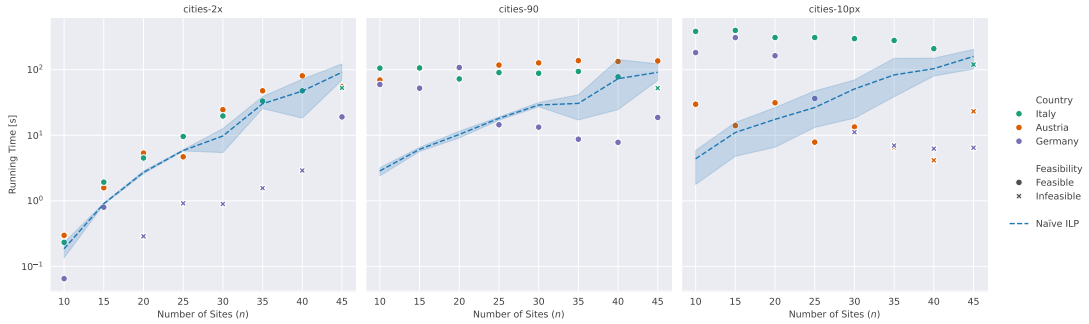
(a) (b)

Figure 5.5: Labeling of the (a) 25 and (b) 45 largest German cities with 90 ports.

consider instances with ordering constraints. As the running time for Naïve ILP varies, for a given n, little across the countries, we report in Figure 5.6a for this solver only the average running time together with the range of measured running times. We can observe that the number of ports has a huge impact on the running time, as the increase of the running time with larger n is for cities-90 and cities-10px barely visible. However, every instance can be solved within 400 seconds, i.e., seven minutes. If we vary m with n, we can observe that small to medium-sized instances of up to twenty-five sites can be computed in under ten seconds, and for the larger ones, it takes between one and two minutes.

Figure 5.6a suggests that recognizing an infeasible instance can be done an order of magnitude faster. Therefore, we analyzed the running time of all infeasible instances, including those with ordering constraints. We plot them in Figure 5.6b. As the running time plots from Figure 5.6a already suggest, we can identify infeasible instances comparably fast. Especially when paired with ordering constraints, we can often classify an instance as infeasible within a second. We have already given one possible explanation for this when arguing why instances with ordering constraints are more likely to be infeasible, namely because we would block off many ports. Hence, we must only create and evaluate comparably few instances in the DP-Algorithm until we conclude that the instance is infeasible. For grouping constraints alone, but also when paired with intra-ordering constraints, it takes longer to identify infeasible instances. This is most likely because infeasibility can be detected only after evaluating almost all instances in the algorithm.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) Running time for our solvers on the cities datasets.

(b) Time required to detect infeasible instances in the cities datasets.

Figure 5.6: Running time on various aspects for the cities datasets (log-plots).

Sample Labelings

We provide in Figures 5.7 to 5.9 sample labelings of this dataset. Each figure shows the 25 largest Italian, Austrian, and German cities, respectively, once labeled with our algorithm and once with Naïve ILP. Note that for Germany, the instance in the cities-2x dataset with $n = 25$ is infeasible. Hence, Figure 5.9 contains an instance without a labeling. Grouping constraints are indicated by the colors, where black sites and labels without a colored bar represent sites that are not involved in any constraint as they are the only city in the respective administrative region.

We can observe that the labeling created by our algorithm gives a more consistent appearance, as the administrative regions are labeled contiguously. However, we have visually observable longer leaders, as seen, for example, in Figure 5.9. Furthermore, the amount and distribution of the ports in cities-10px and cities-2x give a better visual appeal than in cities-90. In particular, observe for cities-90 the large boundary in Figure 5.8.

5.3.2 Ports

Next, we continue with analyzing the results for the ports dataset. We present in Table 5.3 the fraction of feasible instances, grouped by country and types of constraints. We can see that only 38.5% of the instances were feasible, which is less than for the cities

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

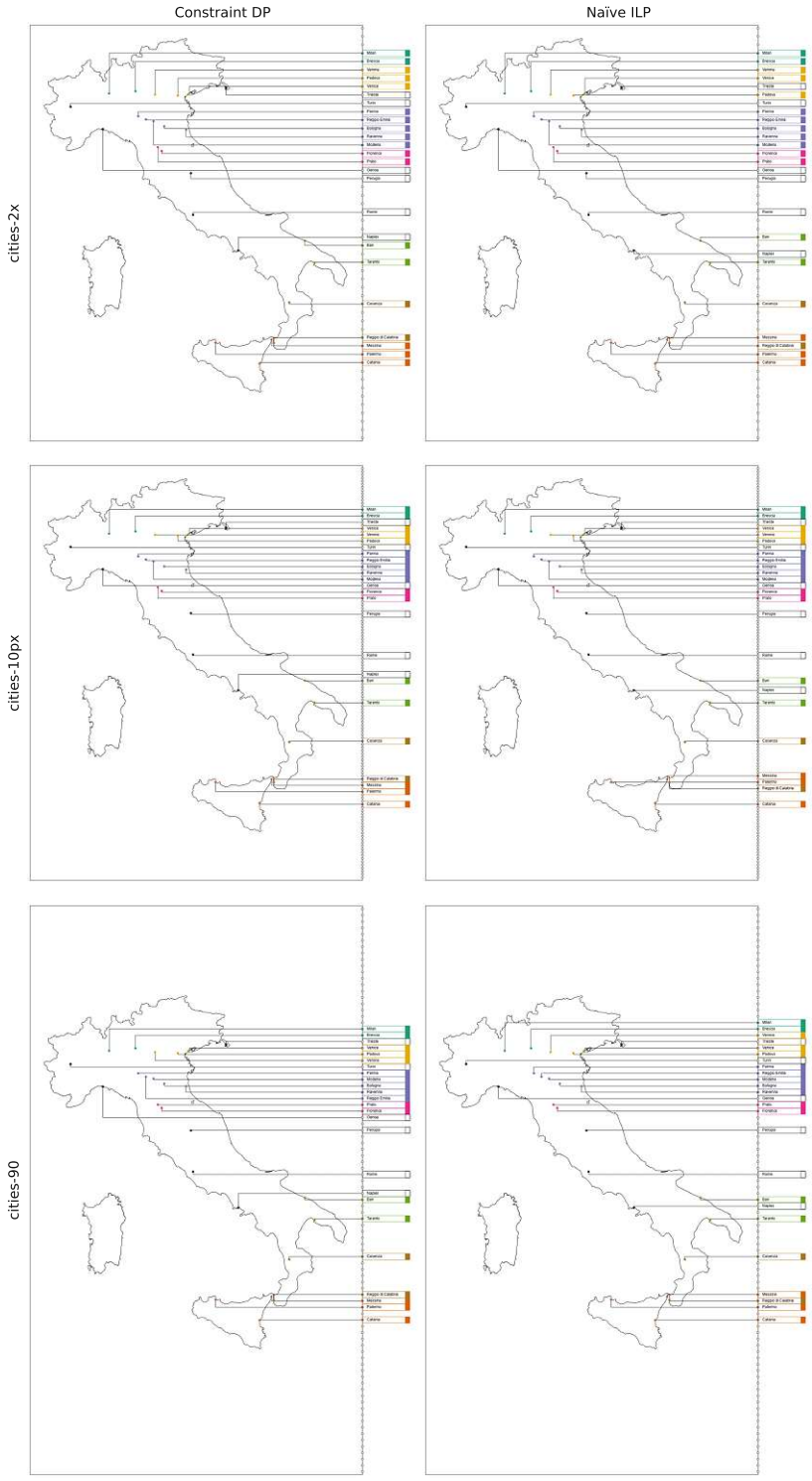


Figure 5.7: Sample labelings of the 25 largest Italian cities from the cities datasets.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

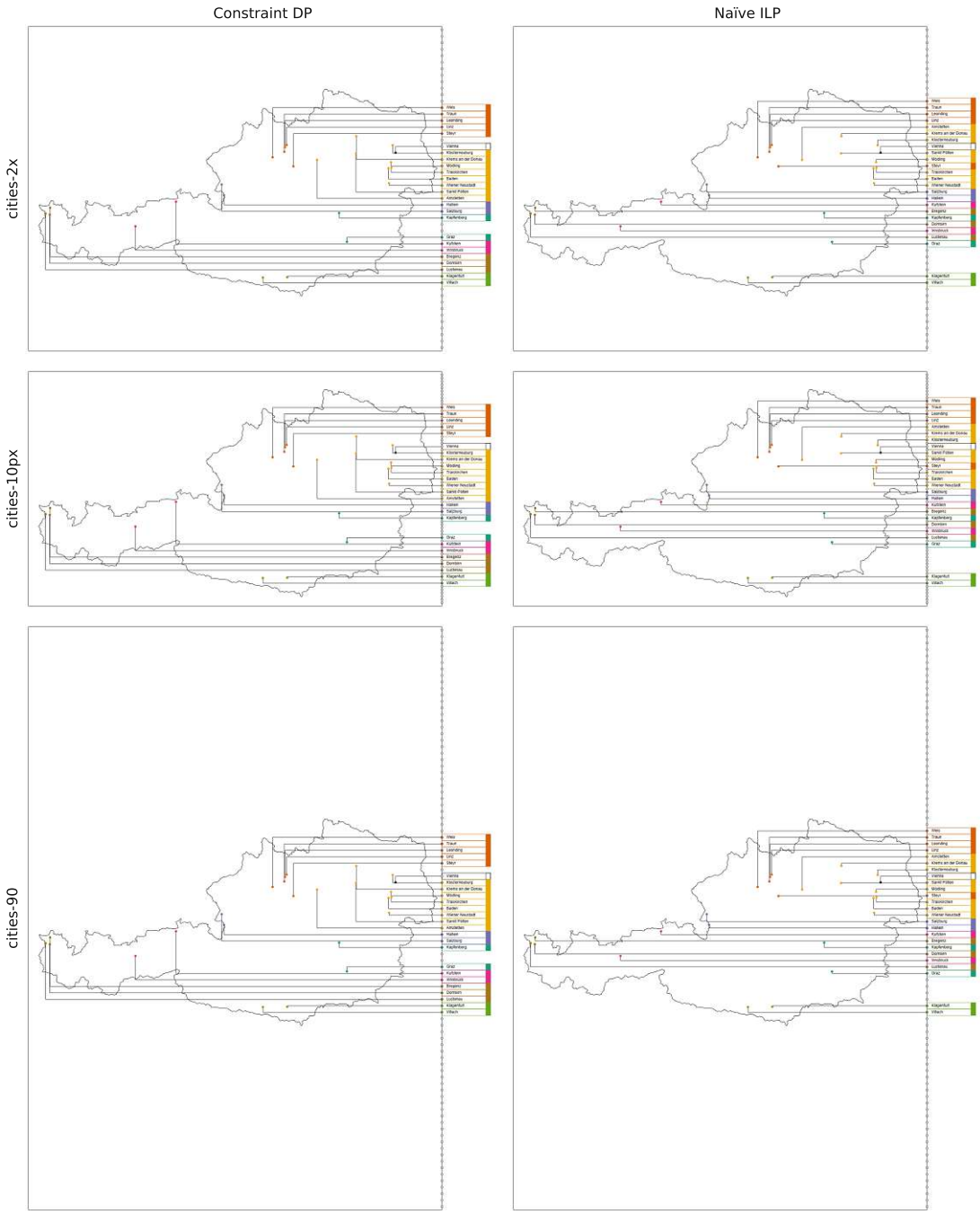


Figure 5.8: Sample labelings of the 25 largest Austrian cities from the cities datasets.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

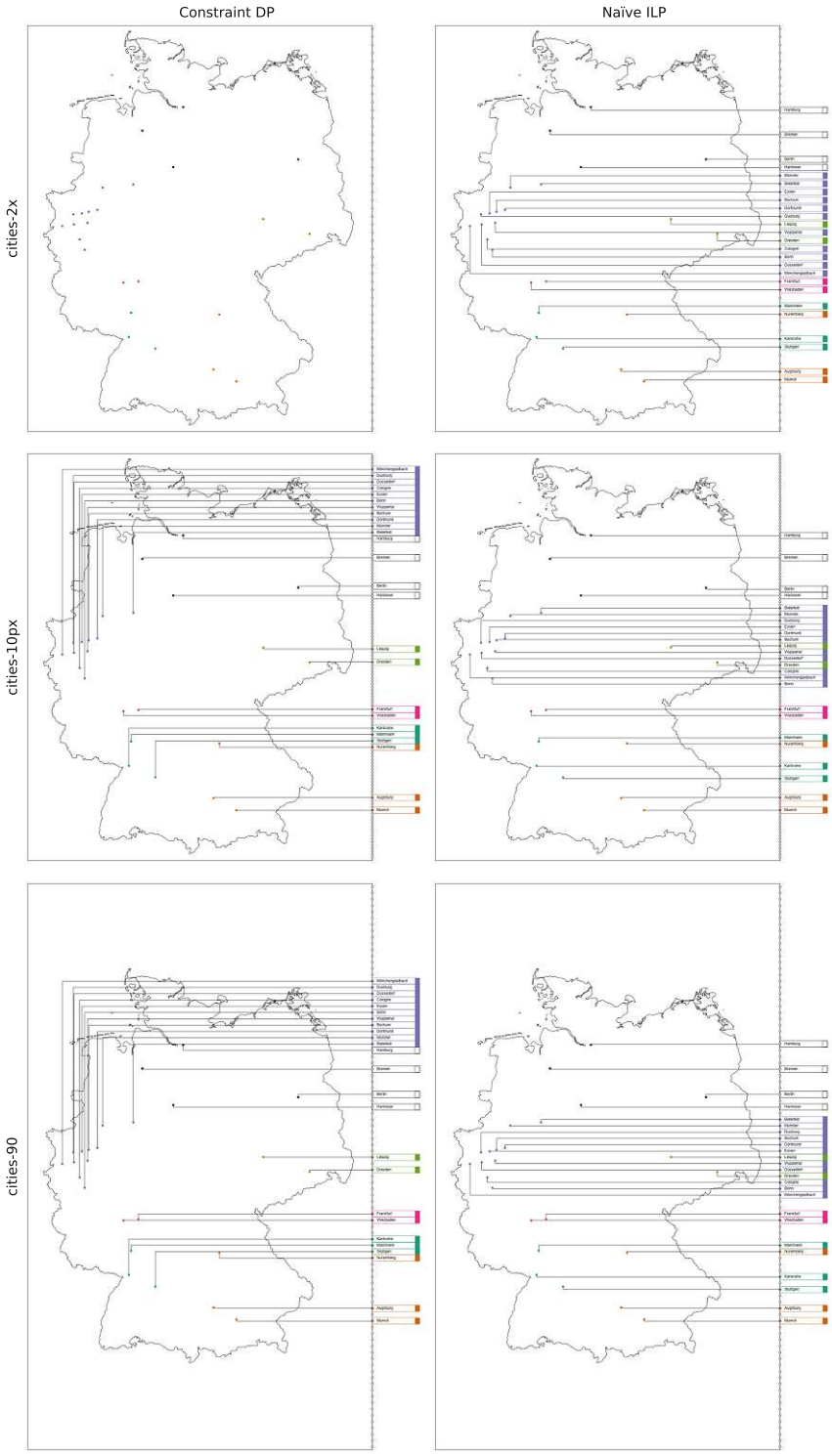


Figure 5.9: Sample labelings of the 25 largest German cities from the cities datasets.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

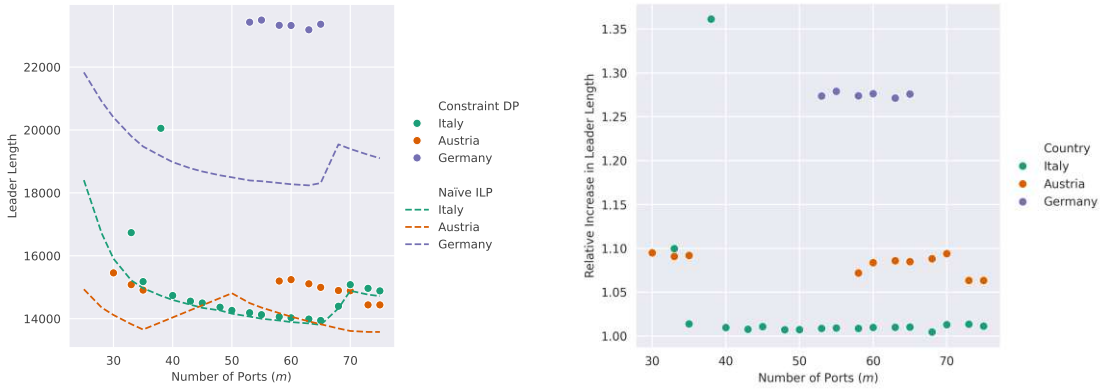
Country	Constraints in the Instances: Gr.*	Intra Or.†	Inter Or.‡	# Feasible / # Total Instances
Austria	✔	✘	✘	11 / 21
	✔	✔	✘	0 / 21
	✔	✘	✔	0 / 21
	✘	✘	✘	16 / 21
Germany	✔	✘	✘	6 / 21
	✔	✔	✘	0 / 21
	✔	✘	✔	0 / 21
	✘	✘	✘	21 / 21
Italy	✔	✘	✘	18 / 21
	✔	✔	✘	4 / 21
	✔	✘	✔	0 / 21
	✘	✘	✘	21 / 21
Austria		–		27 / 84
Germany		–		27 / 84
Italy		–		43 / 84
–	✔	✘	✘	35 / 63
–	✔	✔	✘	4 / 63
–	✔	✘	✔	0 / 63
–	✘	✘	✘	58 / 63
	Total:			**97 / 252**

* Grouping constraints
† Intra-group ordering constraints
‡ Inter-group ordering constraints

Table 5.3: Feasible instances in the ports dataset.

datasets. While we can again observe that almost all instances with ordering constraints are infeasible, here we also have infeasible instances without any constraints. This is because we enforce a minimum distance of ten pixels between two ports, which is half the height of the labels. Hence, although we ensure that there are theoretically, i.e., when compared by number, sufficiently many ports, they can still be too few once we consider their placement. One reason for this is that every other port could be blocked due to labels on the remaining ports. Hence, some instances might already be infeasible without semantic constraints. The arguments we gave for the cities datasets about why ordering constraints tend to be infeasible apply here as well. Furthermore, we compared, where applicable, the leader length and running time of instances with ordering constraints to their counterparts with only grouping constraints. We could observe that, on average, a labeling with ordering constraints has nearly the same leader length as its counterpart,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) Absolute leader length.

(b) Relative increase in leader length for the ports dataset.

Figure 5.10: Comparison of leader lengths obtained with Constraint DP to Naïve ILP for instances from the ports dataset.

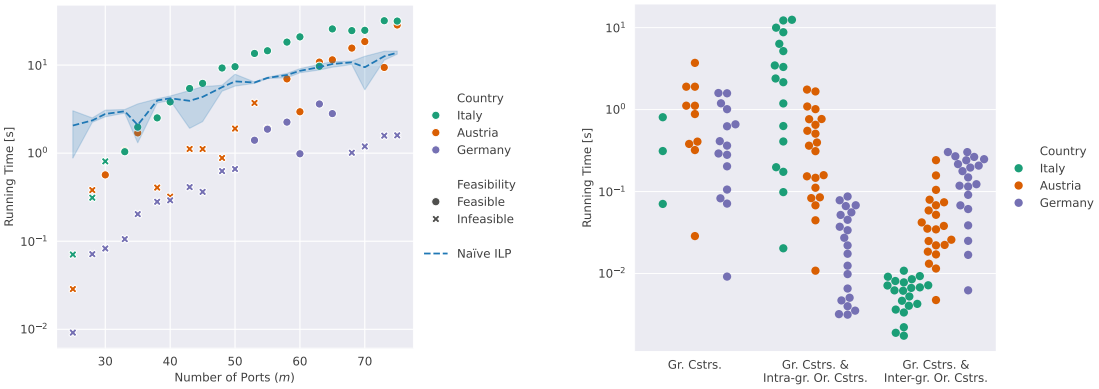
but to compute it, we need 1.16 times the time. As for the cities dataset, this implies that obtaining a feasible labeling that respects the ordering constraints can be done by slightly re-routing the leaders.

Leader Length

We compare in Figure 5.10 the leader length of the feasible instances with grouping constraints to the one computed by Naïve ILP. As for the cities datasets, we can see that the labelings that respect the constraints have longer leaders. We can observe in Figure 5.10a that the leader length decreases with more ports. This was to be expected, as more ports imply that more sites have a port that is in terms of the y-coordinate close by. Hence, we can use shorter leaders to reach them and have to take shorter detours to respect the constraints. Figure 5.10b shows that the relative increase in the leader length is comparable to the cities datasets except for one Italian instance. We can also observe that it varies little with the number of ports. Furthermore, the difference between the solutions from Naïve ILP and Constraint DP is negligible for the Italian cities.

Noteworthy is the increase in the leader length between $m = 35$ and $m = 50$ for Austria and $m = 60$ and $m = 70$ for the other two countries, which can be observed in Figure 5.10a. This is at first counterintuitive, but one possible explanation lies in the positioning of the ports. Recall that we uniformly distribute the ports on the boundary while ensuring a distance of at least ten pixels between two ports. Adding more ports, therefore, alters the position of the existing ports. Hence, a labeling on the previous ports might now be infeasible. Therefore, the labeling could significantly differ from the one on the old ports to ensure feasibility, which explains the increase in leader length.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) Instances with grouping constraints.

(b) Infeasible instances.

Figure 5.11: Running time for our solvers on the ports dataset (log-plots).

Running Time

We have already observed when evaluating the cities datasets that the number of ports m is the dominant factor in the running time. Figure 5.11 gives further insight into how it affects the running time. In particular, we highlight in Figure 5.11a $m = 50$, which is the number of ports for $n = 25$ in Figure 5.6a. As for Figure 5.6a, we aggregate the running times of Naïve ILP and show the mean and the extent of the measured values. In Figure 5.11a, we can observe that for up to $m = 40$, Constraint DP is faster than Naïve ILP, and for larger instances, the running time of both solvers is comparable.

When comparing Figure 5.11a with Figure 5.6a, we can make two interesting observations. On the one hand, the runtime does not seem to increase continuously with the number of ports but decreases several times as we add more ports. On the other hand, the feasibility of the instances is not monotone. For example, the instances with the Austrian cities become feasible with $m = 30$ but then turn infeasible at around $m = 38$ before becoming feasible again at around $m = 58$. One possible explanation for both phenomena is once more the position of the ports. Together with what we have observed for the leader length, we can conclude that the problem and our algorithm are not stable concerning the position of the ports in terms of feasibility, leader length, and running time.

Figure 5.11b shows the running time of infeasible instances. The plot confirms that we can recognize many infeasible instances within a few seconds.

Sample Labelings

We provide in Figure 5.12 sample labelings of this dataset for some instances with Italian and Austrian cities. Since many instances with German cities were infeasible, we refrain from plotting them in the interest of space. The figure shows the labelings computed by our solvers for different numbers of ports. We can see how the placement of the ports, which we mentioned in our discussion several times, can have a huge impact on

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

	Constraints in the Instance		· / #Total Instances	
Sites (n)	Group. Cstrs.*	Ord. Cstrs.†	# Respectable	# Feasible
–	✔	✘	34 / 50	28 / 50
–	✘	✔	44 / 50	32 / 50
–	✔	✔	56 / 100	20 / 100
15	–		25 / 40	14 / 40
20	–		28 / 40	18 / 40
25	–		28 / 40	17 / 40
30	–		27 / 40	16 / 40
35	–		26 / 40	15 / 40
Total:			**134 / 200**	**80 / 200**

* Grouping constraints
† Ordering constraints

Table 5.4: Respectable and feasible instances in the random dataset.

the computed labeling. Compare, for example, in Figure 5.12 the labelings for the Italian cities for $m = 35$, i.e., $1.4n$, with those for $m = 60$, i.e., $2.4n$. While those for $m = 35$ have a more scattered appearance, those for $m = 60$ seem reasonable and compact. Finally, note how in this dataset many of the grouping constraints are (to a large extent) already respected by Naïve ILP, although it does not take them into account when computing the labeling.

5.3.3 Random

As described in Section 5.2.1, we created this dataset with artificial data to see how runtime and feasibility are affected by the distribution of sites and constraints. Hence, we will limit our evaluation of this dataset to these two aspects.

Table 5.4 presents the fraction of respectable and feasible instances in this dataset. From Table 5.4, we can see that 40% of the instances were feasible. In particular, many instances were infeasible when combining grouping and ordering constraints. Table 5.4 shows that many infeasible instances contain constraints that cannot be respected at all. While ordering constraints alone tend to be respectable, when combined with grouping constraints, only 56% of the instances are respectable. One reason for this could be that it is more likely to create cycles among the groups, which results in the constraints being no longer respectable. Hence, in conclusion, we can say that randomly adding constraints that do not arise from the actual semantics of the sites can easily lead to infeasible instances, at least with the given set of ports.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

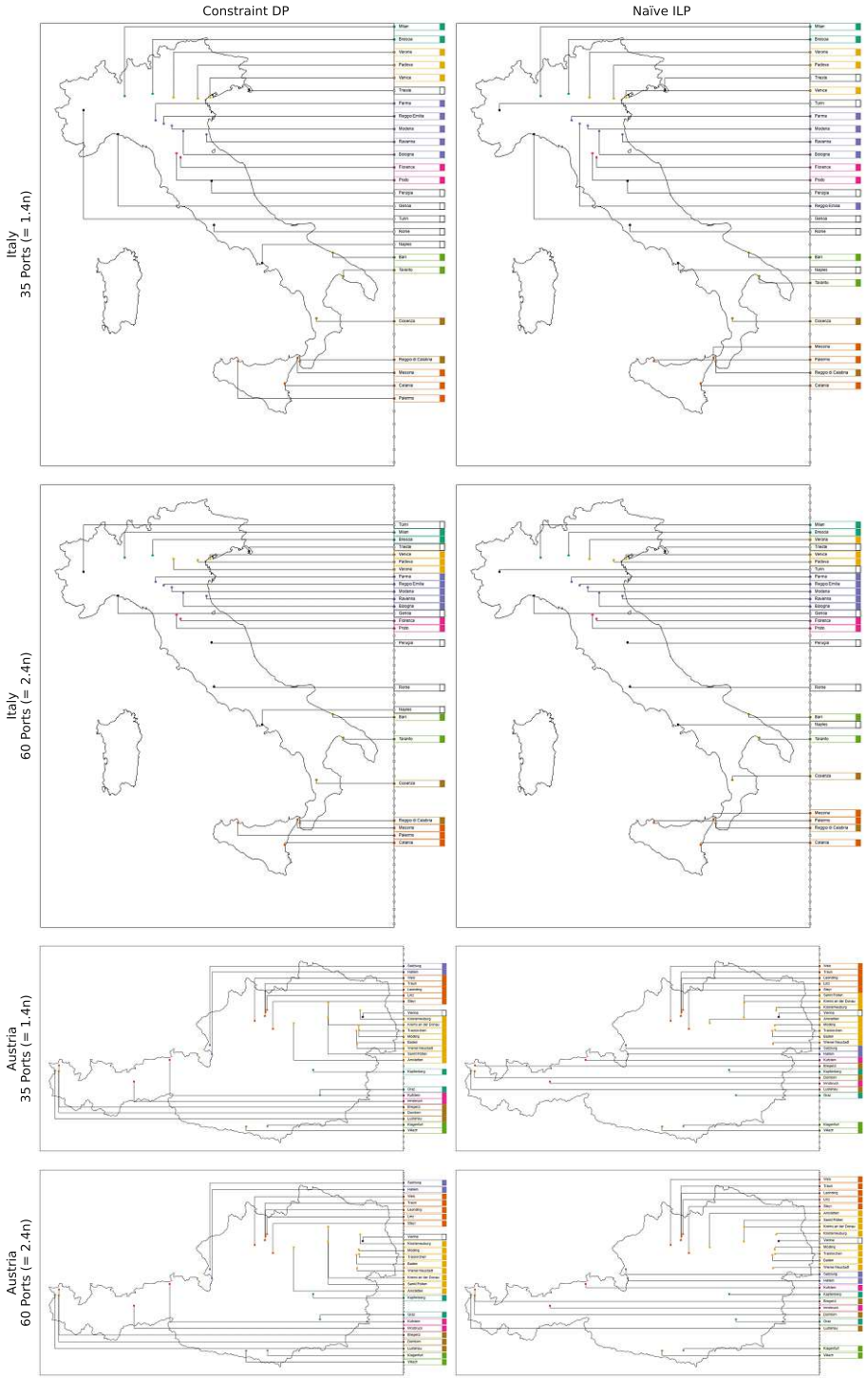


Figure 5.12: Sample labelings for the $n = 25$ largest Italian and Austrian cities with $m = 1.4n$ and $m = 2.4n$ ports from the ports dataset.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

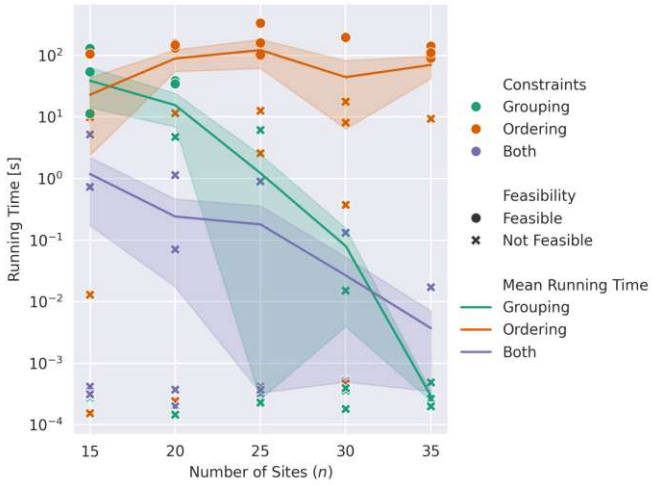
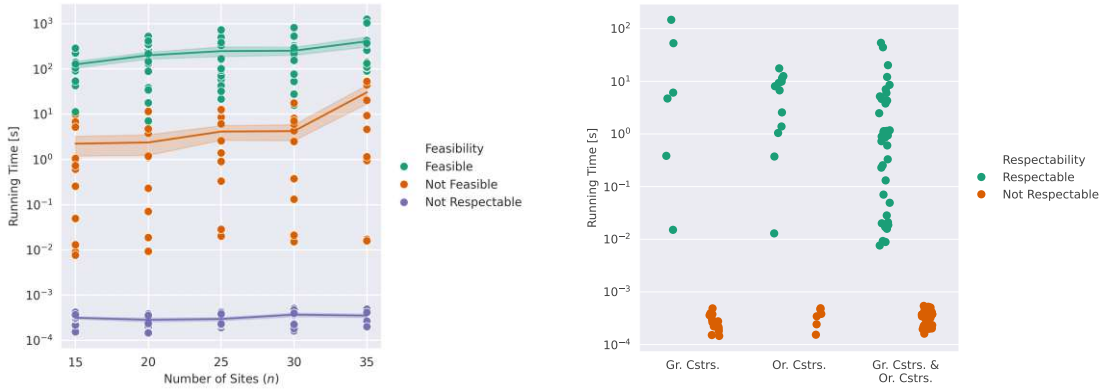


Figure 5.13: Running time of our DP-Algorithm on the random dataset (log-plot).

Running Time

Figure 5.13 shows the running times of a few instances. To maintain readability, we explicitly refrained from presenting all running times and decided to plot the running times of the most constrained instances. As in the previous plots, we indicate with the markers whether the instance was feasible or not. Since we generate per combination of n, k, and r five different instances, we present also the mean and the standard error across the five instances. Apart from the observations we already made for the previous datasets, we can notice two things. First, detecting infeasibility for instances with only ordering constraints seems to take longer than having only grouping constraints. A possible reason for this is that we can only violate an ordering constraint $s \prec s'$ if the leader for the leftmost site uses a port between s and s' or if we label s or s'. However, for (large) grouping constraints, a violation can be detected earlier since separating any two sites of the group will immediately be detected as a violation. Second, the difference in the running times between feasible and infeasible instances is considerably larger in this dataset. We will investigate the latter observation with the help of Figure 5.14. In Figure 5.14a, we plot the mean running time of the instances grouped by their respectability and feasibility together with the standard error. As we can see, the running time for feasible instances is relatively constant, as it is dominated by the number of ports, which remains constant in this dataset. For infeasible instances with respectable constraints, we can observe a relatively high running time with a more noticeable standard error, which can be traced back to the different points in time when we recognize infeasibility. Some instances turn out to be infeasible early, thus having a comparably low running time, whereas for others, especially for those with only ordering constraints, as we have seen in Figure 5.13, detecting infeasibility takes almost as long as computing the labeling. Finally, we can notice that detecting trivial infeasible instances, i.e., instances that contain non-respectable constraints, can be done within milliseconds, as we have the linear-time

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) Running times of feasible, only respectable, and non-respectable constraints.

(b) Running times of infeasible instances.

Figure 5.14: Detailed analysis of the running times for instances from the random dataset (log-plots).

preprocessing routine. That detecting instances with non-respectable constraints can be done considerably faster is also the main conclusion of Figure 5.14b. There, we plot the time required to detect an infeasible instance. We can see that the time needed to detect infeasibility due to non-respectable constraints is almost independent of the number of sites, types of constraints, or their number. The running time for detecting an infeasible instance with respectable constraints, on the other hand, varies considerably among the instances, as the more scattered observations in Figure 5.14b show.

5.3.4 Human Anatomy

Last but not least, we analyze the results of the human anatomy dataset. All of the instances were feasible. Since the main focus of this dataset was to check whether we can label real-world instances on an off-the-shelf laptop, we will mainly analyze the time required to compute the labeling. Nevertheless, we also present the computed labelings.

Running Time

Table 5.5 contains the running time for Constraint DP and Naïve ILP, respectively. We can see that both solvers can find a leader-length-minimal labeling in around five seconds for all instances together. Remarkable is that Constraint DP solves every instance except the largest one faster than Naïve ILP. We consider this an improvement in the daily work of a human illustrator, as they usually need thirty minutes for a single illustration [NNR17]. Furthermore, the labeling created by our algorithm respects the constraints, whereas the one by Naïve ILP does not, as our sample labelings of Figure 5.15 show. Finally, we want to point out that, as expected, we spend almost the entire time in our algorithm on filling the DP-Table. Every other part of the algorithm, such as

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

		Constraint DP	
Instance	Naïve ILP	Total	Of Which Bookkeeping*
Fig. 8.30	0.7922	0.6084	0.0003
Fig. 8.81	0.4295	0.3700	0.0003
Fig. 9.23	0.5023	0.3658	0.0004
Fig. 12.33	0.2519	0.0418	0.0003
Fig. 12.59	2.4632	3.6388	0.0006

* This includes everything but filling the DP-Table.

Table 5.5: Running times of our algorithms on the human anatomy dataset in seconds.

creating the PQ-A-Graph, ensuring that the constraints are respectable, or retrieving the final labeling, was done in a few milliseconds.

Sample Labelings

Figure 5.15 shows the labelings created by Constraint DP and Naïve ILP. We visualize the ordering constraints with the arcs embedded next to the labels. Note that we have overlapping grouping constraints. Therefore, Figure 5.15 does not visualize all constraints. In particular, there is a grouping constraint in the instance *Fig. 12.33* among the orange sites and the uppermost purple site. Therefore, the black site must be labeled below the purple ones to not violate this constraint. We notice that Naïve ILP never respects all constraints. Furthermore, we can observe that the labeling computed by Constraint DP deviates in the majority of the cases only slightly from the one computed by Naïve ILP.

5.3.5 Overview and Final Verdict on the Experiments

Our discussion of the results gave us important insights into the problem and conclusions for our algorithm, which we want to summarize in this section.

First, a pattern we observed throughout the different datasets is that many instances are infeasible in the presence of ordering constraints. Comparisons with instances without ordering constraints revealed that feasible instances have a negligible higher leader length. Hence, it seems that ordering constraints are only suited for enforcing locally limited orders but not suited to putting labels for sites far away into relation. Furthermore, we could observe that instances based on geographic locations were, also without ordering constraints, sometimes infeasible. Since the instances based on the Sobotta atlas of human anatomy [WP13] were feasible, we can conclude that mindlessly adding restrictions without considering the geometric position of the sites and ports is not sensible. However, adding semantic constraints can be fruitful if an illustrator has already an idea of what they want to visualize. As we have seen, looking only for a leader-length-minimal labeling can (partly) respect, by coincidence, some of the semantic constraints, but there is no guarantee for that. It also turned out that our algorithm is for small to medium-sized

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

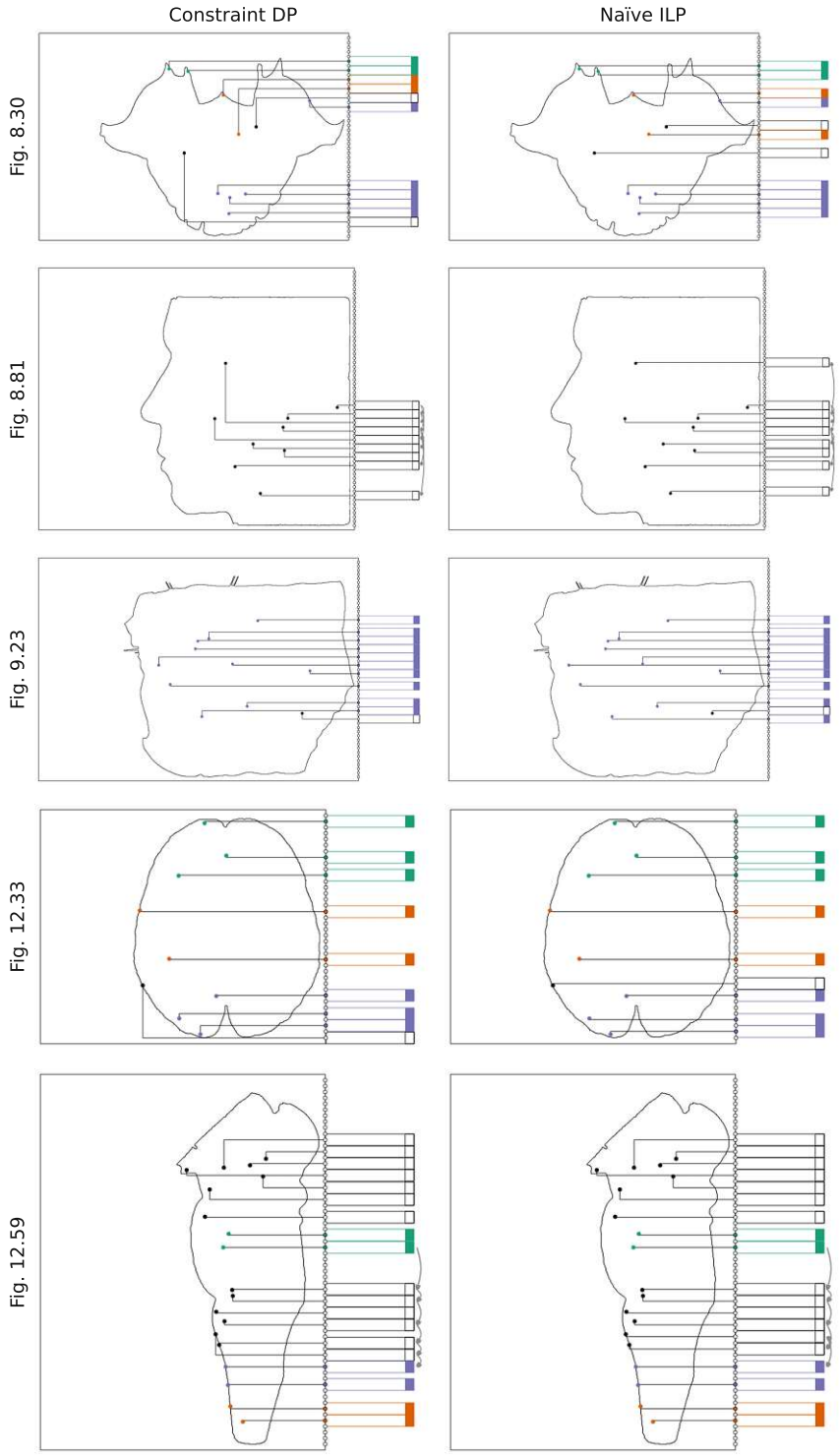


Figure 5.15: Sample labelings of the instances from the human anatomy dataset.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

instances fast enough to compute a labeling in a few seconds or classify them as infeasible. On the other hand, if the instance is large, the running time can be seen as a limitation of our approach, especially in practical applications. One reason for this was that the running time seems, on the practical side, to be more dependent on the number of ports than on the number of sites. We could also observe that the quality and feasibility of the labeling strongly depend on the ports, which makes considering a setting without a pre-determined set of ports interesting and relevant.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Variations of 1-Sided Constrained Boundary Labeling

Throughout this thesis, we made different assumptions about our model and decisions on how to interpret grouping and ordering constraints. While some of them turned out to be crucial for efficient algorithms, others served purely to give us a starting direction. It is time to look into variations that arise from the 1-Sided Constrained Boundary Labeling problem once we no longer have these assumptions or change them.

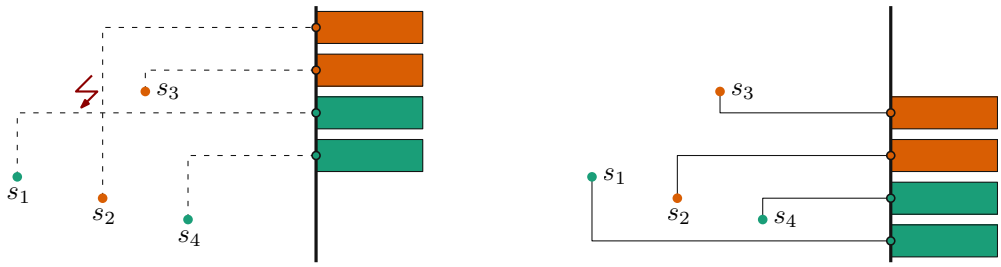
We start with Section 6.1, where we no longer assume that the candidate ports are part of the input. Similar to Fink and Suri [FS16], we observe that we can consider a restricted set of candidate label positions and re-use our dynamic programming algorithm from Chapter 4. In Section 6.2, we neglect ordering constraints and assume that the grouping constraints induce a partition on the sites since this often occurs in real-world examples. Finally, in Section 6.3, we allow the violation of grouping or ordering constraints but have to compensate for this in the optimization function.

We see this chapter only as a first look into some variations that might arise. Therefore, we leave it open for future work to further analyze the proposed variations or define new extensions of b-Sided Constrained Boundary Labeling.

6.1 Sliding Reference Points

Until now, we could place each label at one of m different candidate positions. The main task of the algorithm that we introduced in Chapter 4 was, therefore, to select for each site $s \in \mathcal{S}$ one single port $p \in \mathcal{P}$ out of that m ports, such that the resulting labeling fulfills all of our constraints, while at the same time being optimal with respect to some quality criterion. Having only m different placements for the labels has some algorithmic advantages. Observe that we have for n sites and m ports up to $\binom{m}{n}n! = m^{\underline{n}}$, i.e., m to the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) This instance does not possess a feasible labeling on the given reference points.

(b) The instance from Figure 6.1a that we can label with an alternative set of reference points.

Figure 6.1: An instance whose feasibility depends on the position of the reference points.

n falling, different labelings one would need to try. Although there are exponentially many combinations, thanks to the observation that the leader of the leftmost site subdivides an instance, we can enumerate all relevant site-port combinations in time polynomial in the size of the instance. In particular, recall that the recurrence relation we used in our DP-Algorithm of Chapter 4 does not work if we do not have finitely many label candidates for a site. Hence, it seems reasonable to assume that those reference points are part of the instance. However, this simplistic approach has two disadvantages. On the one hand, it requires that a human illustrator determines suitable label positions by hand and potentially re-runs the algorithm several times with different candidate positions until they are satisfied with the result. On the other hand, while we can find an optimal solution given the reference points, there is no guarantee that this will be the overall best solution, i.e., when neglecting the predefined reference points.

To circumvent these limitations, the literature proposed models and algorithms that do not assume a set of reference points as input. Noteworthy in this regard is the initial work by Nöllenburg et al. [NPS10] and the follow-up results by Huang et al. [HPL14].

The motivations for sliding reference points we gave above already apply to the general boundary labeling problem, i.e., where we do not have any semantic constraints. However, specifying a fixed set of potential label positions in the presence of grouping or ordering constraints has a further disadvantage that is arguably more severe. Consider, for example, the instance from Figure 6.1, together with the grouping constraints $\mathscr{G} = \{\{s_1, s_4\}, \{s_2, s_3\}\}$, which we also indicated with the colors of the sites in the figure. Observe that the instance from Figure 6.1a does not possess a feasible labeling. No matter where we label the green group, their leaders either enclose an orange site, as in Figure 6.1a, or cut off a port that would be required by the orange sites. However, after moving the reference points downwards as shown in Figure 6.1b, we can find a feasible labeling. This peculiarity that, in the presence of our semantic constraints, the feasibility of an instance also depends on the placement of the reference points makes algorithms for flexible label positions inevitable.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.1.1 The 1-Sided Constrained Boundary Sliding Labeling Problem

We now extensively discussed why it is sometimes necessary to abandon the pre-defined set of reference points and allow the labels to be placed everywhere along one side of the boundary, as long as the resulting labeling is feasible. We can easily model this new freedom of the label placement by slightly altering the definition of 1-Sided Constrained Boundary Labeling (see Problem 3).

Problem 7 (1-Sided Constrained Boundary Sliding Labeling)**.**

Given: *A set of n sites $\mathcal{S} = \{s_1, \ldots, s_n\}$, an infinite vertical line ν to the right of $\mathcal{S}$, constraints $\mathcal{C} = (\mathcal{G}, \prec)$ consisting of a family of k grouping constraints $\mathcal{G}$ and a relation $\prec$ consisting of r ordering constraints, a label height $h > 0$, and a computable function $f : \mathcal{S} \times \mathbb{R} \to \mathbb{R}_0^+$.*

Task: *Find a feasible 1-sided po-labeling $\mathcal{L}$ that respects $\mathcal{C}$, minimizes $f(\mathcal{L})$, and in which all labels have their port on ν.*

Note that we still require that the port of a label is at its (left) vertical center. Observe that the problem definition allows to place the labels anywhere on the infinite vertical line ν. However, in real-world applications, this is not always possible. One usually then asks to place the labels within a given (vertical) interval that represents, for example, the available height on a book page. Nevertheless, to simplify arguments, we assume that there is no such restriction on the placement of the labels but see it as an interesting open problem.

In the following section, we will show that we do not have to consider all (infinite) label positions on the entire (unbounded) vertical line but can restrict ourselves to a set of polynomially many candidate ports. To that end, we will show this result for the most common optimization functions, feasibility, leader-bend-minimization, and leader-length-minimization. We see investigating other optimization functions as a direction for further research.

6.1.2 Reducing to 1-Sided Constrained Boundary Labeling by Defining a Suitable Finite Set of Reference Points

While 1-Sided Constrained Boundary Sliding Labeling seems notoriously difficult, we will show in this section that it is sufficient to consider only $\mathcal{O}(n^2)$ many reference points. This result will use Assumption 6, which states that the vertical distance between any pair of sites must not be a multiple of the label height h. We want to point out that Assumption 6 is not a real restriction since we can achieve this by moving some sites slightly.

Assumption 6. *The vertical distance $|y(s) - y(s')|$ between two sites $s, s' \in \mathcal{S}$, $s \neq s'$, must not be a multiple of h, the height of the labels.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

In the following, we first restrict ourselves to finding some feasible labeling. Later, we provide arguments to adapt the proof to find a leader-bend-minimal or leader-length-minimal feasible labeling. As in the literature, we will call a set of touching labels a *stack* of labels [NPS10].

Checking on the Feasibility of an Instance

Let $\mathcal{I}$ be an instance of 1-Sided Constrained Boundary Sliding Labeling that satisfies Assumption 6. We define the following.

$$d := \min_{\substack{s,s' \in \mathcal{S}, \\ s \neq s'}} \left(|y(s) - y(s')| - qh \right), \text{ where } q = \left\lfloor \frac{|y(s) - y(s')|}{h} \right\rfloor$$

Note that d equals the smallest value one would need to add to a multiple of h such that there are two sites in $\mathcal{S}$ that are this far apart. Due to Assumption 6, $d > 0$ holds. Let $0 < \varepsilon < d$ be an arbitrary but instance-dependent constant. We create the following set $\mathcal{P}(\mathcal{S})$ of ports. For each port $p \in \mathcal{P}(\mathcal{S})$, we have $x(p) = x(\nu)$. Hence, we only state their y-coordinates.

$$\mathcal{P}(s) := \{y(s) + qh,\ y(s) + qh + \varepsilon,\ y(s) - qh,\ y(s) - qh - \varepsilon \mid 0 \leq q \leq n\}$$
$$\mathcal{P}(\mathcal{S}) := \bigcup_{s \in \mathcal{S}} \mathcal{P}(s)$$

Clearly, $|\mathcal{P}(\mathcal{S})| = \mathcal{O}(n^2)$. Furthermore, the definition of d gives rise to the following observation.

Observation 6.1. *For each $s, s' \in \mathcal{S}$, $s \neq s'$, there exists a $p \in \mathcal{P}(s)$ such that we have $|y(s) - y(p)| < |y(s) - y(s')|$, i.e., between any two sites there is at least one port.*

The following Lemma 6.1 builds on top of Observation 6.1 to show that it suffices to consider only ports from $\mathcal{P}(\mathcal{S})$ to check whether an instance $\mathcal{I}$ has a feasible labeling.

Lemma 6.1. *Let $\mathcal{I}$ be an instance of* 1-Sided Constrained Boundary Sliding Labeling *that satisfies Assumption 6. If $\mathcal{I}$ possesses a feasible labeling, then there also exists one in which each port is from $\mathcal{P}(\mathcal{S})$.*

Proof. Our proof builds on arguments used by Fink and Suri for a similar result [FS16, Lemma 1].

Let $\mathcal{L}$ be a feasible labeling of $\mathcal{I}$ in which, for the sake of the proof, not all ports are from $\mathcal{P}(\mathcal{S})$. We will transform it into a feasible labeling $\mathcal{L}'$ in which each port is from $\mathcal{P}(\mathcal{S})$. Note that throughout the proof we will never change the order of the labels. Hence, if $\mathcal{L}$ respects the constraints, so does $\mathcal{L}'$. Let s_t and s_b be the top-most and bottom-most sites in the instance, respectively. If there are labels that have its port above $y(s_t) + \varepsilon$,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

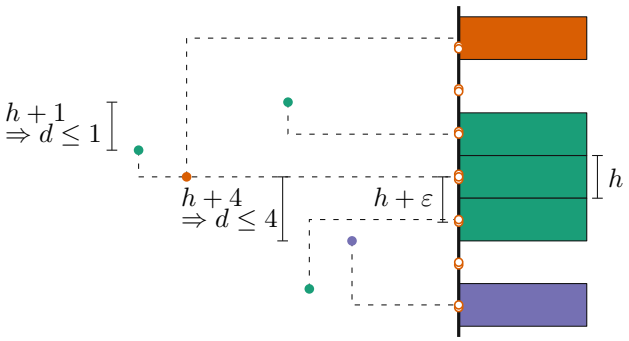


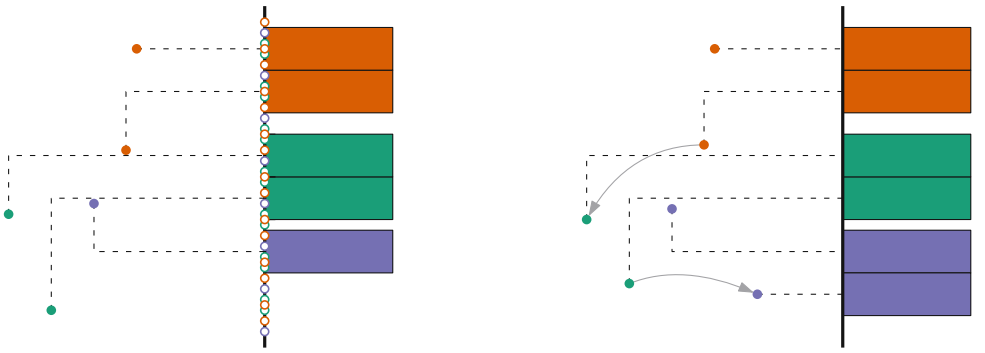
Figure 6.2: The leader of a green site crosses the orange site after moving all stacks upwards. We only show the ports that arise from the orange site.

we move all of them simultaneously down until the bottom-most one, let it be ℓ, either hits the port $p \in \mathcal{P}(s_t)$ with $y(p) = y(s_t) + \varepsilon$, or another label. In either case, we no longer move ℓ, but continue the same process with the other labels above s_t. Observe that in the former case, ℓ is positioned at a port from $\mathcal{P}(\mathcal{S})$. Furthermore, the label that is the next one above ℓ will eventually hit it and is thus also placed at a port from $\mathcal{P}(\mathcal{S})$. We do the same with labels that are entirely below s_b. However, we move them upwards. Since these labels were entirely above s_t or below s_b, and also remain that, this cannot introduce any crossings.

Then, for the remaining labels not yet positioned at a port from $\mathcal{P}(\mathcal{S})$, we proceed bottom-to-top as follows. We take the bottom-most not yet positioned label ℓ and move it upwards until it either is positioned at a port from $\mathcal{P}(\mathcal{S})$ or hits another label ℓ'. In the former case, we stop. In the latter case, ℓ might not yet be at a port from $\mathcal{P}(\mathcal{S})$. To ensure the feasibility of $\mathcal{L}'$, we "merge" those labels into a stack and move from now on this entire stack and thus all its labels simultaneously. Observe that ℓ can never move past by a site since any site induces a port in $\mathcal{P}(\mathcal{S})$, i.e., in the worst case, we stop at a $p \in \mathcal{P}(\mathcal{S})$ with $y(p) = y(s)$ for some $s \in \mathcal{S}$. This may result in leaders crossing other sites, but we will address that later. First, we continue to move all the not-yet-positioned labels in the same manner.

In the end, we have to deal with the aforementioned leader-site crossings. Such crossings can arise if in $\mathcal{L}$ a leader λ, originated at a site s, passes between a port of $\mathcal{P}(\mathcal{S})$ and a site $s' \in \mathcal{S}$, $s \neq s'$. If we then move labels upwards, we may first hit the port induced by s', i.e., the port p with $y(p) = y(s')$, before any other port. Depending on the position of s', i.e., if $x(s) \leq x(s')$ holds, λ now crosses s'. To resolve this crossing, we take that label and the stack it belongs to, if there is one, and move it downwards until we hit a port from $\mathcal{P}(\mathcal{S})$. By our selection of ε, it is guaranteed that we will hit a port before we hit another site since there is at least one other port strictly between the end of the stack and any other site s'', as $0 < \varepsilon < d \leq \left(|y(s') - y(s'')| - \left\lfloor \frac{|y(s')-y(s'')|}{h} \right\rfloor h\right)$ holds, and we have a port at $y(s') - \varepsilon$ and one at $y(s'') + \varepsilon$. Figure 6.2 also visualizes this with the orange ports. Note that $d \leq 1$ holds in Figure 6.2. Therefore, we have $\varepsilon < 1$ and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) The labels are trapped between two sites if we do not include the ε in the definition of the ports.

(b) This instance with sliding reference points possesses a feasible but no leader-length-minimal feasible labeling.

Figure 6.3: Instances that show (a) why we need the ε in the discretization step and (b) how an instance might have no leader-length-minimal labeling.

consequently $h + \varepsilon < h + d \leq h + 4$, the distance between the orange and the purple sites. If we move all such (stacks of) labels simultaneously, we can never overrun a placed label by moving downwards but might merge with another stack already placed at ports from $\mathcal{P}(\mathcal{S})$. Now, in the resulting labeling $\mathcal{L}'$, each label is located at a port $p \in \mathcal{P}(\mathcal{S})$, and since we never changed the order of the labels when transforming $\mathcal{L}$ into $\mathcal{L}'$, all constraints are still respected. □

We get the following corollary from $|\mathcal{P}(\mathcal{S})| = \mathcal{O}(n^2)$.

Corollary 6.1 (Together with Theorem 4.1)**.** *Let $\mathcal{I} = (\mathcal{S}, \nu, \mathcal{C} = (\mathcal{G}, \prec), h, f)$ be an instance of* 1-Sided Constrained Boundary Sliding Labeling *that satisfies Assumption 6 with $n = |\mathcal{S}|$, $k = |\mathcal{G}|$, and $r = |\prec|$. We can compute a feasible labeling of $\mathcal{I}$ in $\mathcal{O}(n^{11} \log n + k + \|\mathcal{G}\|)$ time using $\mathcal{O}(n^6)$ space if such a labeling exists.*

In contrast to Fink and Suri [FS16], we require the additional ε in the definition of the ports. The reason for this is to ensure planarity without changing the order of the labels while moving them. Therefore, we have to ensure that a stack of labels is not enclosed between two sites without having the ability to reach a port without intersecting a site, as in Figure 6.3a. In the proof by Fink and Suri, we can resolve the situation of Figure 6.3a by moving the green stack of labels up (or down) until we hit a site with our leader and then exchange the respective leaders and labels. However, this is impossible in our setting, as this would violate the green grouping constraint.

Minimizing Leader Bends

If we seek a leader-bend-minimal labeling, we first recall the optimization function that we defined in Section 2.4.1 and note that it only matters whether a site s is labeled at the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

(single) port p with $y(p) = y(s)$, if it exists. Any other port induces an identical cost in the labeling. Now, observe that $\mathcal{P}(s)$ introduces such a port for each $s \in \mathcal{S}$. Since $\mathcal{P}(\mathcal{S})$, therefore, contains for each site such a port, we conclude, with the help of Lemma 6.1, that a leader-bend-minimal labeling of an instance $\mathcal{I}$ of 1-Sided Constrained Boundary Sliding Labeling must exist on our discretized set of ports $\mathcal{P}(\mathcal{S})$ if $\mathcal{I}$ possesses a feasible labeling at all. Corollary 6.2 summarizes this conclusion.

Corollary 6.2 (Using results of Theorem 4.1)**.** *Let $\mathcal{I} = (\mathcal{S}, \nu, \mathcal{C} = (\mathscr{G}, \prec), h, f)$ be an instance of* 1-Sided Constrained Boundary Sliding Labeling *that satisfies Assumption 6 with $n = |\mathcal{S}|$, $k = |\mathscr{G}|$, and $r = |\prec|$. We can compute a leader-bend-minimal labeling of $\mathcal{I}$ in $\mathcal{O}(n^{11} \log n + k + \|\mathscr{G}\|)$ time using $\mathcal{O}(n^6)$ space if such a labeling exists.*

Minimizing Leader Length

In contrast to leader-bend-minimal labelings, retrieving leader-length-minimal labelings for sliding reference points is not so trivial. One reason for this is that not every instance that possesses a feasible labeling also possesses a leader-length-minimal feasible labeling, as Figure 6.3b shows. There, the geometric position of the sites, together with the grouping constraints expressed by the colored sites, and the ordering constraints expressed by the arcs, force us to order the groups of labels as shown in the figure. Now assume that a feasible labeling $\mathcal{L}^*$, for example, the one in Figure 6.3b, would be leader-length-minimal. Since $\mathcal{L}^*$ is feasible, there is no site-leader crossing. But no matter how close the leader of the green sites passes by the purple site, we can always halve the vertical distance between the site and the leader. This results in a feasible labeling with shorter leaders, a contradiction to the optimality of $\mathcal{L}^*$.

To circumvent this limitation, we enforce that any leader must maintain a vertical distance of $d_{\min} > 0$ to other sites. As a consequence, we no longer work with the ε but define a new set of ports $\mathcal{P}(\mathcal{S})$. Recall that for each port $p \in \mathcal{P}(\mathcal{S})$, we have $x(p) = x(\nu)$. Hence, we only state the y-coordinates of the ports in $\mathcal{P}(\mathcal{S})$.

$$\mathcal{P}(s) := \{y(s) + qh,\ y(s) + qh + d_{\min},\ y(s) - qh,\ y(s) - qh - d_{\min} \mid 0 \leq q \leq n\}$$
$$\mathcal{P}(\mathcal{S}) := \bigcup_{s \in \mathcal{S}} \mathcal{P}(s)$$

Clearly, we still have $|\mathcal{P}(\mathcal{S})| = \mathcal{O}(n^2)$. Although we have changed the set of ports, we can still show that they are sufficient for finding a leader-length-minimal labeling.

Lemma 6.2. *Let $\mathcal{I}$ be an instance of* 1-Sided Constrained Boundary Sliding Labeling *that satisfies Assumption 6. If $\mathcal{I}$ possesses a feasible labeling $\mathcal{L}$ where each leader maintains a vertical distance of at least $d_{\min}$ to other sites, then there exists a feasible labeling $\mathcal{L}'$ of $\mathcal{I}$ in which each port is from $\mathcal{P}(\mathcal{S})$. Furthermore, we have $f(\mathcal{L}') \leq f(\mathcal{L})$, where f measures the leader length of a labeling.*

Proof. The crux of the proof is similar to the one for Lemma 6.1. However, note that in the initial labeling $\mathcal{L}$, all leaders maintain a vertical distance of at least $d_{\min}$ to other

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

sites. Since we place a port this far away from each site s, we can never hit a site with our leaders while moving (stacks of) labels, i.e., we can never introduce a site-leader crossing.

The remainder of the proof is identical to Lemma 6.1, except that we move (stacks of) labels always in a non-increasing direction with respect to the leader lengths. This means that if in a stack more labels have their site above the respective port, we move the stack upwards, and vice-versa. We break ties arbitrarily. As already observed by Fink and Suri [FS16], with this operation we never increase the overall leader length. Thus, for the labeling $\mathcal{L}'$ that we eventually obtain with this operation, we have $f(\mathcal{L}') \leq f(\mathcal{L})$. □

Note that $\mathcal{P}(\mathcal{S})$ can contain ports that would lead to an infeasible labeling if certain sites are labeled there due to not maintaining a vertical distance of at least $d_{\min}$. However, observe that in the proof of Lemma 6.2, we never moved past a port. Since we started with a feasible labeling that maintains the distance $d_{\min}$ and created ports that are that far away from the sites, this can never result in a labeling violating the vertical distance.

Corollary 6.3 (Together with Theorem 4.1)**.** *Let $\mathcal{I} = (\mathcal{S}, \nu, \mathcal{C} = (\mathscr{G}, \prec), h, f)$ be an instance of* 1-Sided Constrained Boundary Sliding Labeling *that satisfies Assumption 6 with $n = |\mathcal{S}|$, $k = |\mathscr{G}|$, and $r = |\prec|$. We can compute a leader-length-minimal labeling of $\mathcal{I}$ where each leader maintains a vertical distance of at least $d_{\min}$ to other sites in $\mathcal{O}(n^{11} \log n + k + \|\mathscr{G}\|)$ time using $\mathcal{O}(n^6)$ space if such a labeling exists.*

6.2 Disjoint Grouping and No Ordering Constraints

Although we showed in Theorem 4.1 that we can solve 1-Sided Constrained Boundary Labeling in polynomial time, the result is arguably only from little practical relevance due to the high running time. As we have seen in Chapter 5, computing a leader-length-minimal labeling can take for larger instances several minutes. Especially if one recalls that Benkert et al. [BHKN09] solved 1-Sided Boundary Labeling in $\mathcal{O}(n^2m^3)$ time, assuming that the optimization function can be evaluated in constant time, we can expect a significant difference in the running times.

One reason for the high running time is the intrinsic checks to ensure that a candidate port is feasible for the leftmost site. While our PQ-A-Graph can represent all respectable constraints, including complex intersecting combinations of grouping constraints, this generality is not always required. Many real-world instances have only simple grouping constraints and completely lack ordering constraints. For example, in atlases of human anatomy, there are often only a few grouping constraints per figure, explicitly indicated by curly brackets, and most of the time they enclose only a few sites [WP13]. Furthermore, many grouping constraints from the real world partition the set of sites. Consider, for example, the administrative regions of a country. They induce a partition of the cities, as every city of that country belongs to exactly one region.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

We can assume without loss of generality that each site is in a grouping constraint, as we can always introduce singleton groups. Due to the above observations, it seems, from a practical point of view, reasonable to work under Assumption 7 in some cases.

Assumption 7. *The groups in $\mathscr{G}$ form a partition of $\mathcal{S}$, i.e., for $\mathscr{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$, it holds $\mathcal{G}_i \cap \mathcal{G}_j = \emptyset$, $1 \leq i, j \leq k$, $i \neq j$, and $\mathcal{S} = \bigcup_{i=1}^{k} \mathcal{G}_i$. There are no ordering constraints, i.e., $\prec = \emptyset$.*

Assumption 7 can also be summarized as not having any ordering enforced on the sites, neither explicitly with the ordering constraints $\prec$ nor implicitly, due to intersecting grouping constraints.

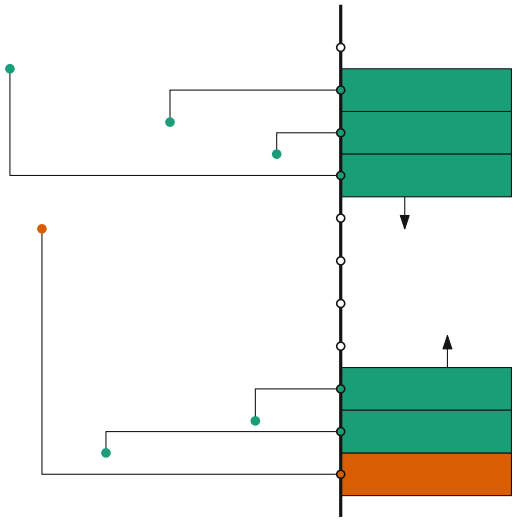
In this section, we will exploit this assumption and use it to show that we can assume that any group is continuous, i.e., without an unused reference point in between. As this follows the ideas behind a stack of labels with sliding reference points, we call such a set of labels a *stack* of labels. We want to remind the reader that we use port as a synonym for reference point unless stated otherwise.

Lemma 6.3. *Let $\mathcal{I}$ be an instance of* 1-Sided Constrained Boundary Labeling *that satisfies Assumption 7. If $\mathcal{I}$ possesses a feasible labeling, then there exists also a labeling of $\mathcal{I}$ in which the labels of each group form a stack.*

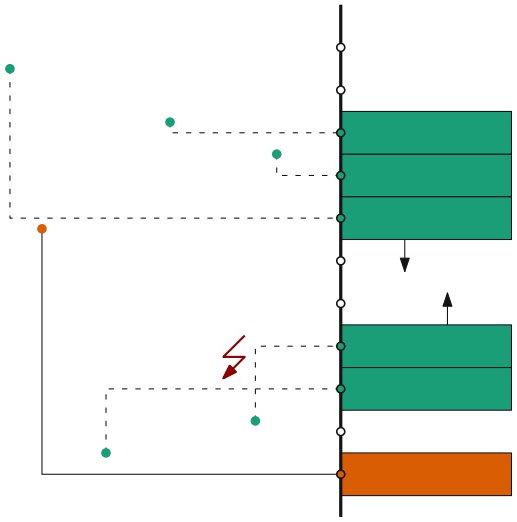
Proof. The proof is constructive, and we refer to Figure 6.4 for illustrations that support some arguments of the proof. Let $\mathcal{L}$ be a labeling of the instance $\mathcal{I}$ in which there exists at least one group $\mathcal{G} \in \mathscr{G}$ whose labels do not form a stack. We will adapt $\mathcal{L}$ to a new feasible labeling $\mathcal{L}'$ in which all groups that previously formed a stack still do so, and in addition, the labels for $\mathcal{G}$ also do form a stack. By repeatedly applying this procedure, we eventually arrive at a labeling in which the labels for each group form a single stack.

We assume, without loss of generality, that the labels for the group $\mathcal{G}$ form two stacks. If they form more than two stacks, we can iteratively take any two stacks next to each other and apply the transformations below until the labels for $\mathcal{G}$ eventually form a single stack. As $\mathcal{L}$ is a feasible labeling, there cannot be a label between the two stacks of $\mathcal{G}$, i.e., they only have unoccupied ports between each other, and we have a situation as in Figure 6.4a. We start moving the two stacks of labels closer to each other by moving them one port at a time. We either stop when (i) they are next to each other and we can merge them into a single stack, or (ii) we would run with the leader of a site in the group over another site $s \in \mathcal{S}$. In the former case, we are done. In the latter case, we have to distinguish whether s is part of the group. If $s \in \mathcal{G}$ holds, i.e., the site is part of the group, then we exchange the labels, as shown in Figures 6.4b and 6.4c, and continue moving the stacks closer together. However, if $s \notin \mathcal{G}$, we have to be more careful. In the following, we assume, without loss of generality, that s would be overrun by the top stack that moves down, as shown in Figure 6.4d. The other case, i.e., where the bottom stack overruns s while moving upwards, is symmetric. If we would overrun s with the

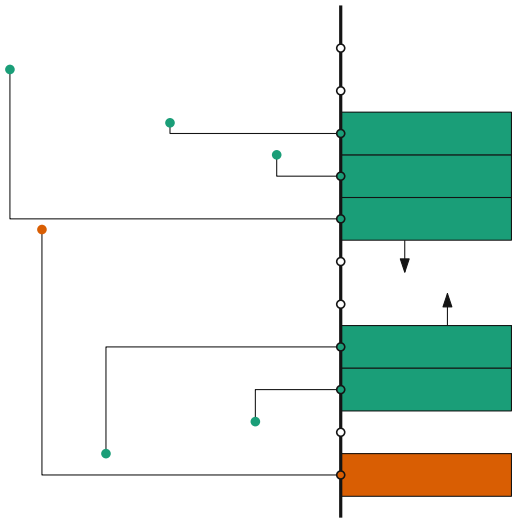
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



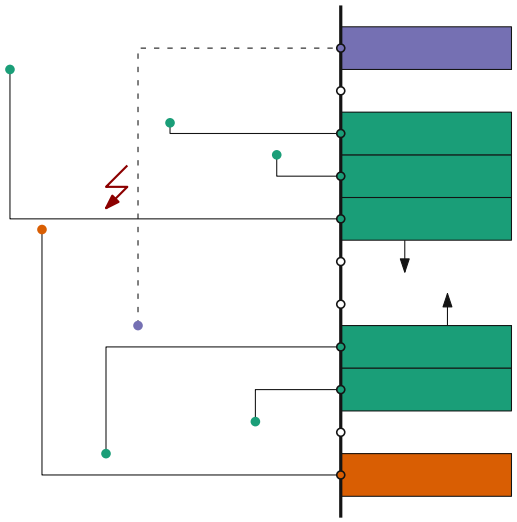
(a) The two green stacks move towards each other.



(b) The lower stack overruns a leader of a site that belongs to the same group, which results in a leader-crossing.



(c) To make the labeling from Figure 6.4b planar, we re-route the involved leaders.



(d) If both green stacks would overrun sites not part of the group, then the original labeling must have been non-planar.

Figure 6.4: Figures illustrating the arguments of the proof for Lemma 6.3.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

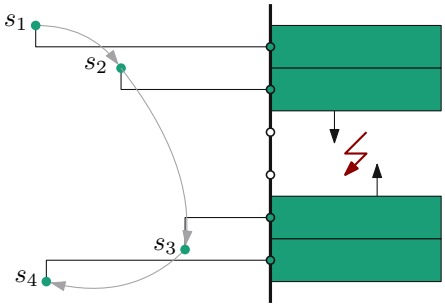


Figure 6.5: If this instance violates Assumption 7 by, for example, having the partial order $\prec = \{s_1 \prec s_2, s_2 \prec s_3, s_3 \prec s_4\}$, as illustrated, or the grouping constraints $\mathscr{G} = \{\{s_1, s_2, s_3, s_4\}, \{s_1, s_2\}, \{s_2, s_3\}, \{s_3, s_4\}\}$, then there does not exist a feasible labeling where all groups form a single stack.

top stack while moving down, we stop moving with the top stack. We observe that the port for the leader λ of s is below s. This is because $\mathcal{L}$ is feasible and hence s has to be labeled either above or below $\mathcal{G}$. Therefore, λ must entirely pass by at least one of the stacks of $\mathcal{G}$. However, since we would overrun s with a leader while moving the top stack down, λ cannot pass the upper stack of the labels for $\mathcal{G}$ without crossing that leader. Therefore, λ must go downwards, and we cannot move the top stack further without introducing a crossing. We proceed with moving the bottom stack upwards until we merge with the top stack. Observe that it is guaranteed that while doing so, we can never overrun another site $s' \notin \mathcal{G}$, $s \neq s'$, as in Figure 6.4d, with a leader. Because by similar arguments as before, the leader λ' of s' must go upwards to a feasible position for the label of s'. However, if the leader λ of s runs downwards to pass the bottom stack of labels and the leader λ' of s' upwards to pass the top stack of labels, and one of s and s' is more to the right, this inevitably leads to a crossing with one of the leaders for $\mathcal{G}$ that would overrun s or s', which we also illustrate in Figure 6.4d. This would contradict the feasibility of the labeling $\mathcal{L}$. Therefore, we can conclude that if we hit a site that is not part of the group while moving the top stack downwards, we know that we can move the bottom stack all the way upwards until we eventually merge the two stacks into a single one for $\mathcal{G}$. □

Observe how the proof of Lemma 6.3 uses the fact that we can interchange the order of the labels within a stack. If this would not be possible due to, for example overlapping grouping constraints or explicit orders on the sites, i.e., if we violate Assumption 7, then some instances possess a feasible labeling, but not one where all labels from a group form a single stack, for example, as in Figure 6.5.

6.2.1 The Collapsed Pyramid Structure

Assumption 7 implies that we can assume that the labels for the sites of each group form a single stack. If we consider the stack for a group $\mathcal{G} \in \mathscr{G}$, we can observe that the leader

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

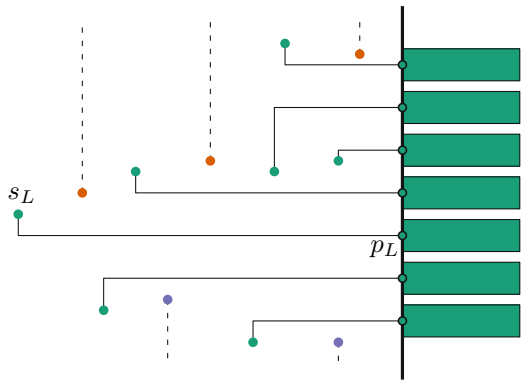


Figure 6.6: The collapsed pyramid for the green group of sites.

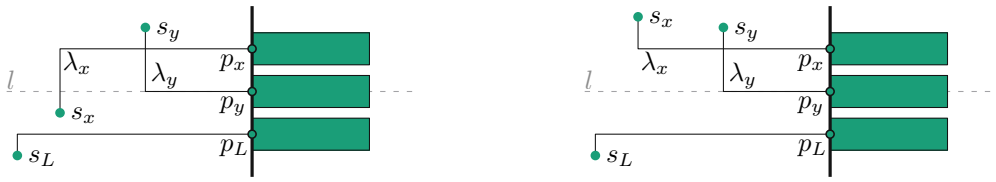
of the leftmost site s_L of $\mathcal{G}$ splits the group into three parts, $\mathcal{G}^a$ and $\mathcal{G}^b$, consisting of the sites **a**bove and **b**elow the leader of s_L, respectively, and s_L itself. To ensure that sites in $\mathcal{G}$ with a long horizontal segment in their leader, i.e., far to the left, do interfere, or rather block, as few sites as possible, they should be labeled as close to s_L as possible. In a naïve way, one would label the remaining sites of $\mathcal{G}$ "from left to right" to avoid any overlaps with sites outside of $\mathcal{G}$. However, this is not always possible, as we could "cut off" sites of $\mathcal{G}$ located below a port. Nevertheless, we can follow this intuition as long as possible. This results in a labeling structure that we call the collapsed pyramid and define in the following. Let $\mathcal{G}$ be a group with its leftmost site $s_L \in \mathcal{G}$ being labeled at $p_L \in \mathcal{P}$. This partitions $\mathcal{G} \setminus \{s_L\}$ into $\mathcal{G}^a$ and $\mathcal{G}^b$. For $\mathcal{G}^a$, create two lists, $L_L = \mathcal{G}^a$ and $L_R = \emptyset$. Sweep a line l from p_L upwards. Whenever l hits a site $s \in \mathcal{G}^a$, remove s from L_L and add it to L_R. Whenever l hits a port p, let s be the rightmost site in L_R; if L_R is empty, then the leftmost site in L_L. Label s at p and repeat this until all sites in $\mathcal{G}^a$ are labeled. The structure that results from repeating the symmetric procedure for $\mathcal{G}^b$ is the *collapsed pyramid*, or simply *pyramid*, for $\mathcal{G}$ with its *foundation* at p_L. Figure 6.6 shows an example of the collapsed pyramid for the group of green sites. s_L builds the foundation of the pyramid at the port p_L. The length of the horizontal segments of the leaders are, outgoing from $\lambda_L = (s_L, p_L)$, in principle non-increasing, and would form a pyramid. We call this the *pillars* of the pyramid. If this would cause crossings, we have to use "shorter" leaders, i.e., label first sites further to the right. We say that the pyramid *collapses* at such sites. Some of the pillars in the upper half of the pyramid from Figure 6.6 are collapsed. Note that the collapsed pyramid does not necessarily result in a leader-length-minimal labeling.

In the following, we prove some basic properties of the collapsed pyramid.

Lemma 6.4. *The collapsed pyramid for a group $\mathcal{G} \in \mathscr{G}$ with the foundation at p_L does not have any crossing* among *leaders* from sites of $\mathcal{G}$.

Proof. Let $\mathcal{L}$ be a labeling that contains the collapsed pyramid for the group $\mathcal{G} \in \mathscr{G}$. In the following, we concentrate on $\mathcal{G}^a$, as the leaders for a site in $\mathcal{G}^a$ and in $\mathcal{G}^b$ or the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) s_x is below p_y. (b) s_x is above p_y.

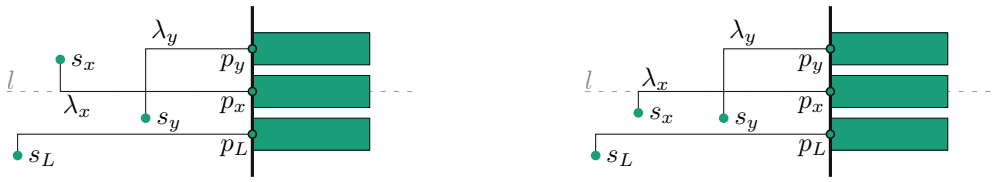
Figure 6.7: Figures illustrating the arguments of the proof for Lemma 6.4 – Cases where λ_x is above λ_y.

leftmost site s_L in $\mathcal{G}$ trivially never intersect. Furthermore, we consider only sites from $\mathcal{G}^a$ as the arguments for sites from $\mathcal{G}^b$ are symmetric. Towards a contradiction, assume that in $\mathcal{L}$, there would be a crossing among the leaders of sites from $\mathcal{G}^a$, let it be s_x and s_y. Assume that s_y is further to the right than s_x. Hence, the crossing must involve the vertical segment of the leader $\lambda_y = (s_y, p_y)$ of s_y and the horizontal segment of the leader $\lambda_x = (s_x, p_x)$ of s_x. We differentiate whether (the horizontal segment of) λ_x is above or below (the one of) λ_y.

λ_x is above λ_y. If λ_x runs above λ_y, as in Figure 6.7, then the port p_x for s_x is above the port p_y for s_y. Furthermore, we can see that λ_y must run downwards, i.e., s_y is above p_y. However, s_x can be above or below p_y. For s_x being below p_y, when the sweep line l was at p_y (gray dashed line in Figure 6.7a), we have that s_x is in the list L_R and s_y in the list L_L. Since this means that L_R is non-empty, we would have never selected s_y in this situation – Contradiction to the definition of $\mathcal{L}$. If s_x is above p_y, as in Figure 6.7b, then, when our procedure to create $\mathcal{L}$ was at p_y (gray dashed line in Figure 6.7b), we have that s_x and s_y are in the list L_L. However, as s_x is further to the left than s_y, we would not have selected s_y in this situation – Contradiction to the definition of $\mathcal{L}$. We can see that if λ_x runs above λ_y, then we always arrive at a contradiction.

λ_x is below λ_y. When λ_x runs below λ_y, as in Figure 6.8, then the port p_x for s_x is below the port p_y for s_y. Furthermore, we can see that λ_y must run upwards, i.e., s_y is below p_y. However, s_x can be either above or below p_x, and we will differentiate between these scenarios. If s_x is above p_x, as in Figure 6.8a, and we hit with the sweep line l p_x (gray dashed line in Figure 6.8a), we have that s_x is in L_L and s_y is in the list L_R. Similar to above, this means that L_R is non-empty, and we would have never selected an entry from L_L. Hence, we would not have selected s_x in this situation – Contradiction to the definition of $\mathcal{L}$. If s_x is below p_x, as in Figure 6.8b, then, when our procedure to create $\mathcal{L}$ was at p_x (gray dashed line in Figure 6.8b), we have that both s_x and s_y are in L_R. However, since this implies that L_R is non-empty, we would take the rightmost entry of L_R. As s_y is further to the right than s_x, we would have never selected s_x – Contradiction to the definition of $\mathcal{L}$. We can see that also when λ_x runs below λ_y, we arrive at a contradiction.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) s_x is above p_x. (b) s_x is below p_x.

Figure 6.8: Figures illustrating the arguments of the proof for Lemma 6.4 – Cases where λ_x is below λ_y.

As the above cases are exhaustive and always lead to a contradiction, we can conclude that there cannot be a crossing within $\mathcal{G}$, if $\mathcal{G}$ was labeled as a collapsed pyramid. □

While Lemma 6.4 guarantees that the collapsed pyramid is free of crossings within its group, Lemma 6.5 will ensure that no overlaps can occur outside the group, provided that a feasible labeling exists at all.

Lemma 6.5. *Let $\mathcal{I}$ be an instance of* 1-Sided Constrained Boundary Labeling *that satisfies Assumption 7. If $\mathcal{I}$ possesses a feasible labeling, then there also exists a feasible labeling of $\mathcal{I}$ in which each group is labeled as a collapsed pyramid.*

Proof. For an instance $\mathcal{I}$, satisfying Assumption 7, let $\mathcal{L}$ be a labeling that witnesses the feasibility of $\mathcal{I}$. Due to Lemma 6.3, we can assume without loss of generality that each group in $\mathcal{L}$ forms a single stack. Let $\mathcal{G} \in \mathscr{G}$ be a group whose labels do not form the collapsed pyramid. This proof will be constructive, and we show that we can transform the feasible labeling $\mathcal{L}$ into a feasible labeling $\mathcal{L}'$ where the labels for $\mathcal{G}$ form a single stack as a collapsed pyramid. Repeating the argument for all groups proves the lemma. Let s_L be the leftmost site in $\mathcal{G}$ and p_L its port in $\mathcal{L}$. We define $\mathcal{L}'$ to be the labeling where we replace the labeling of $\mathcal{G}$ with the collapsed pyramid with the foundation at p_L but leave all other labels unchanged. Note that in $\mathcal{L}$ and $\mathcal{L}'$, the labels for $\mathcal{G}$ form a stack. Hence, we occupy in both labelings the same ports for $\mathcal{G}$. As a consequence, $\mathcal{L}'$ must respect all grouping constraints as $\mathcal{L}$ did so. In the following, we show that $\mathcal{L}'$ is still crossing-free. Since by Lemma 6.4 we know that there cannot be a crossing among the leaders for sites in $\mathcal{G}$, the only crossings that could occur are with leaders for sites outside $\mathcal{G}$. We continue using proof by contradiction and assume that a leader for a site $s \in \mathcal{G}$ intersects with a leader for a site $s' \notin \mathcal{G}$. Without loss of generality, we assume that $s \in \mathcal{G}^a$ holds. Since we did not change the foundation of the pyramid, s and s' must be labeled above $\lambda_L = (s_L, p_L)$ in $\mathcal{L}$ and $\mathcal{L}'$.

Let s and s' be labeled in $\mathcal{L}'$ at the ports p and p', respectively. Depending on the position of s and s' with respect to p, i.e., above or below the port, and to each other, i.e., which of s and s' is further to the left, we arrive at eight different cases.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

We can neglect all four cases where s' is left of s. As we do not change the port where s' is labeled, either s' would split the group $\mathcal{G}$ in $\mathcal{L}$, for example, if s is below p', or $\mathcal{L}$ is non-planar, for example, if s is above p'. However, this contradicts our assumptions on $\mathcal{L}$.

From the four cases where s is left of s', only the two remain where s' is below p, as the other two imply that $\mathcal{L}$ would not respect the grouping constraint $\mathcal{G}$. Observe that s left of s' implies that s' is above λ_L. We continue arguing by cases depending on the position of s with respect to p. See Figure 6.9 for illustrations supporting the following arguments.

***s* is above *p*.** Consider Figure 6.9a for an illustration of the situation where s is above p. Let $\overline{p}$ be the port where we labeled s in $\mathcal{L}$ and observe that $\overline{p}$ must be below p, as $\mathcal{L}$ is feasible. Note that we have $s \in L_L$ at $\overline{p}$. Since in both labelings, $\mathcal{L}$ and $\mathcal{L}'$, the labels for $\mathcal{G}$ form a stack, and p is above $\overline{p}$, there must be some site $\overline{s}$ that we label in $\mathcal{L}'$ at $\overline{p}$ or a port below, but in $\mathcal{L}$ at a port strictly above $\overline{p}$. We can observe the following depending on the position of $\overline{s}$ with respect to $\overline{p}$.

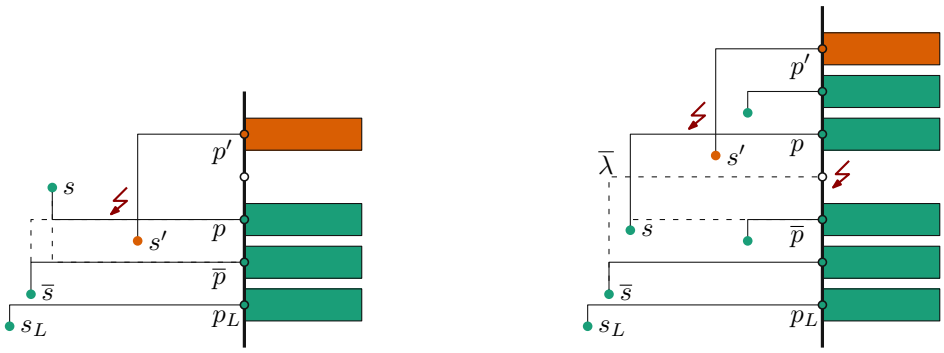
If $\overline{s}$ is above $\overline{p}$, it would be in L_L at $\overline{p}$, assuming we would not remove the site from the list once labeled. This means that $\overline{s}$ must be left of s, as otherwise we would have labeled s further below in $\mathcal{L}'$. However, to ensure that $\mathcal{L}$ is planar, we must label $\overline{s}$ in $\mathcal{L}$ above s. But this means that the leader for $\overline{s}$ crosses the leader for s' in $\mathcal{L}$, or $\overline{s}$ is labeled above s' in $\mathcal{L}$. Both contradict the feasibility of $\mathcal{L}$.

If $\overline{s}$ is below $\overline{p}$, it would be in L_R at $\overline{p}$, as in Figure 6.9a. Therefore, $\overline{s}$ must be left of s, because we labeled $\overline{s}$ in $\mathcal{L}$ above $\overline{p}$. Hence, if $\overline{s}$ would be right of s, then the leaders for s and $\overline{s}$ would cross in $\mathcal{L}$. But even if $\overline{s}$ is left of s, as in Figure 6.9a, the same reasoning as above leads to a contradiction. Therefore, the situation from Figure 6.9a cannot occur.

***s* is below *p*.** Consider now the case where s is below p and see Figure 6.9b for an illustration. Again, let $\overline{p}$ be the port where we labeled s in $\mathcal{L}$ and observe that it must be below p. If $\overline{p}$ is below s, the same arguments as above imply that there is a site $\overline{s}$ that was labeled above s in $\mathcal{L}$. On the other hand, if $\overline{p}$ is above s, then s is in L_R when creating $\mathcal{L}'$. Since we did not label s at $\overline{p}$, but somewhere above $\overline{p}$, there must be some other site in L_R that is further to the right. This site must have been labeled in $\mathcal{L}$ at a port below $\overline{p}$, and as the labels for $\mathcal{G}$ form a stack in both labelings, there must be some other site $\overline{s}$ that is now labeled below $\overline{p}$ but was labeled above $\overline{p}$ in $\mathcal{L}$. As s is below p (and $\overline{p}$), we must label $\overline{s}$ at a port below s to avoid crossings. If $\overline{s}$ was in L_L when we labeled it in $\mathcal{L}'$, then it must be left of s, as also s is in L_L at that port. But even if $\overline{s}$ was in L_R at that port, it must be left of s. Recall that in $\mathcal{L}$, $\overline{s}$ is labeled above $\overline{p}$. Therefore, if $\overline{s}$ is not to the left of s, this would result in crossing leaders in $\mathcal{L}$.

We have now derived that there must be a site $\overline{s}$ left of s that is labeled below $\overline{p}$ in $\mathcal{L}'$ but above this port in $\mathcal{L}$. Due to the reasoning from the case where s was above p, $\mathcal{L}$ cannot label $\overline{s}$ above s' and, therefore, $\overline{s}$ must be labeled between s and s' in $\mathcal{L}$. Consider the topmost site left of s that we label above s in $\mathcal{L}$. The site $\overline{s}$ guarantees its existence,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) s is above p. (b) s is below p.

Figure 6.9: Figures illustrating the arguments of the proof for Lemma 6.5. Dashed lines indicate $\mathcal{L}$, solid lines $\mathcal{L}'$.

as it fulfills these properties. To not introduce another site, we assume that $\overline{s}$ is also the topmost site left of s. However, this simplification is without loss of generality. Since it is left of s and labeled above s in $\mathcal{L}$, all sites, including s, that are right and below it must be labeled below that site in $\mathcal{L}$. In particular, this means there are enough ports to label all those sites and $\overline{s}$. Since some of these sites are now labeled further below, we push some of the remaining labels further up in the $\mathcal{L}'$. However, the leader for $\overline{s}$ shields these sites from the sites labeled above the leader $\overline{\lambda}$ for $\overline{s}$, also depicted in Figure 6.9b. Therefore, we never move these labels further up than the port where $\overline{s}$ is labeled in $\mathcal{L}$. In particular, p is at most the port where $\overline{s}$ was labeled in $\mathcal{L}$. If p is further up, the labels for $\mathcal{G}$ cannot form a stack, as in Figure 6.9b. Since $\mathcal{L}$ is feasible and $\overline{s}$ to the left of s, when we label s at p, its leader can never cross the leader from s'. This, however, contradicts our assumption of $\mathcal{L}'$ having a crossing between the leaders of s and s'. Therefore, the situation from Figure 6.9b can never occur.

Since all cases lead to a contradiction, our initial and only assumption that $\mathcal{L}'$ contains a crossing must have been false. Therefore, if the initial labeling $\mathcal{L}$ was feasible, so is $\mathcal{L}'$. By applying the same arguments several times, we conclude that there exists a feasible labeling of $\mathcal{I}$ where each group forms a collapsed pyramid, if there exists a feasible labeling at all. □

6.2.2 Speeding Up Our Dynamic Programming Algorithm for 1-Sided Constrained Boundary Labeling

Using Lemma 6.5, we can define a DP-Algorithm to compute a feasible labeling of an instance $\mathcal{I}$ of 1-Sided Constrained Boundary Labeling that adheres to Assumption 7.

Theorem 6.1. *Let $\mathcal{I} = (\mathcal{S}, \mathcal{P}, \mathcal{C} = (\mathcal{G}, \emptyset), h, f)$ be an instance of* 1-Sided Constrained Boundary Labeling *that satisfies Assumption 7 with $n = |\mathcal{S}|$, $m = |\mathcal{P}|$, and $k = |\mathcal{G}|$.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

We can check in $\mathcal{O}(n^4m^3 \log m)$ time and $\mathcal{O}(n^2m^2)$ space whether $\mathcal{I}$ possesses a feasible labeling.

Proof. Let $\mathcal{I}$ be an instance of 1-Sided Constrained Boundary Labeling that adheres to Assumption 7. The crux of the proposed algorithm will be to use the generic $\mathcal{O}(n^2m^3)$-time DP-Algorithm blueprint sketched by Bekos et al. [BNN21] for computing 1-sided boundary labelings. However, we exploit Lemma 6.5 and label, instead of the leftmost site, the leftmost group using the collapsed pyramid.

We define a DP-Table D of size k^2m^2 that stores one, if an instance possesses a feasible labeling and zero otherwise. Similar to the DP-Algorithm from Theorem 4.1, we denote with $I = (\mathcal{G}_1, p_1, \mathcal{G}_2, p_2)$ an instance in our DP-Algorithm. However, since we know by Lemma 6.5 that we can assume that in a feasible labeling, every group forms a collapsed pyramid, an instance can be bound by (the leftmost sites of) two groups rather than two sites. Therefore, the semantic of an instance $I = (\mathcal{G}_1, p_1, \mathcal{G}_2, p_2)$ in our DP-Algorithm is that the foundation of the two pyramids for $\mathcal{G}_1$ and $\mathcal{G}_2$ is at p_1 and p_2, respectively. For an instance I let $\mathcal{G}_L \in \mathscr{G}$ be the group to which the leftmost not yet labeled site in the instance belongs. Note that by our definition of the instance, all the sites of $\mathcal{G}_L$ are in I. As in our DP-Algorithm from Theorem 4.1, we denote for an instance I with $\mathcal{S}(\mathrm{I})$ and $\mathcal{P}(\mathrm{I})$ the set of sites and ports, respectively, that are in the instance I. However, note that we now also exclude from these sets the sites and ports already used in a collapsed pyramid that defines the instance.

We say that a port p is a *feasible foundation* for $\mathcal{G}_L$, if we can place the collapsed pyramid for $\mathcal{G}_L$ with the foundation at p without introducing label overlaps. Furthermore, no two sites of another group are split by the pyramid, and no site outside $\mathcal{G}_L$ is surrounded by two pillars of the pyramid for $\mathcal{G}_L$ such that it cannot reach another port without crossing a leader of $\mathcal{G}_L$. If a port is a feasible foundation for $\mathcal{G}_L$, we create the collapsed pyramid for $\mathcal{G}_L$ and recursively evaluate the sub-instances I_1 and I_2, i.e., we use the following relation.

$$D[(\mathcal{G}_1, p_1, \mathcal{G}_2, p_2)] = \max_{\substack{\text{feasible} \\ \text{foundation} \\ p \in \mathcal{P}(I)}} \min\left(D[(\mathcal{G}_1, p_1, \mathcal{G}_L, p)], D[(\mathcal{G}_L, p, \mathcal{G}_2, p_2)]\right) \tag{6.1}$$

Furthermore, we set $D[I] = 1$ for $\mathcal{S}(I) = \emptyset$ and $D[I] = 0$ if we are unable to place a collapsed pyramid for $\mathcal{G}_L$, i.e., if there is no feasible foundation. If $\mathcal{I}$ does possess a feasible solution, we have in the end $D[(\mathcal{G}_0, p_0, \mathcal{G}_{k+1}, p_{m+1})] = 1$, where $\mathcal{G}_0$ and $\mathcal{G}_{k+1}$, and p_0 and p_{m+1} denote dummy groups and ports that bound $\mathcal{I}$ from above and below, respectively. Correctness follows by the correctness of the DP-Algorithm from Theorem 4.1, as Equation (6.1) resembles the recurrence relation used in the DP-Algorithm from Chapter 4, combined with Lemmas 6.3 to 6.5, which guarantee us the existence of a feasible labeling that consists of collapsed pyramids only.

We proceed with analyzing the running time of our DP-Algorithm. As a preprocessing step, we sort all sites and ports in increasing order by their x and y-values, where applicable,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

and assign the sites to the group they belong to. This takes $\mathcal{O}(n \log n + m \log m)$ time. Furthermore, we store for each group $\mathcal{G} \in \mathscr{G}$ the leftmost site that is part of $\mathcal{G}$. As the groups form a partition over the sites, this can be done in $\mathcal{O}(n)$ time by a linear scan over the groups. If we fill the DP-Table D top-down using memoization, we have to evaluate, in the worst case, each possible instance at most once. There are k^2m^2 different instances. We can find the group $\mathcal{G}_L$ in $\mathcal{O}(k)$ time since we store for each group the leftmost site. Once found, we have to evaluate $\mathcal{O}(m)$ different candidate ports p. Determining whether p is a feasible foundation for $\mathcal{G}_L$ includes building the collapsed pyramid structure. To do this, we can locate the leftmost site of $\mathcal{G}_L$ and the port p in the sorted lists in $\mathcal{O}(\log n + \log m)$ time. Then, we can run the sweep line, which takes $\mathcal{O}(|\mathcal{G}_L|)$ time, as we have already sorted the sites and ports. While placing the pillars of the collapsed pyramid, we can continuously check that we do not enclose another site or overlap with another label. This increases the running time of creating the structure to $\mathcal{O}(|\mathcal{G}_L|\, n)$. To ensure that we do not split any two sites that belong to the same group, we can, in the end, naïvely iterate over all $\mathcal{O}(n^2)$ pairs of sites. Overall, this sums up to $\mathcal{O}(\log n + \log m + |\mathcal{G}_L|\, n + n^2) = \mathcal{O}(\log m + n^2)$ time, as we have $|\mathcal{G}_L| \leq n$. Furthermore, as $\mathscr{G}$ forms a partition over $\mathcal{S}$, we have $k = \mathcal{O}(n)$. Combining all, we get a running time of $\mathcal{O}(k^2m^2\,(k + m\,(\log m + n^2))) = \mathcal{O}(n^2m^2\,(n + m\,(\log m + n^2)))$. This can be simplified and upper-bounded by $\mathcal{O}(n^4m^3 \log m)$. Note that this dominates the $\mathcal{O}(n \log n + m \log m)$ running time of the preprocessing steps. The space requirement is trivially in $\mathcal{O}(n^2m^2)$. □

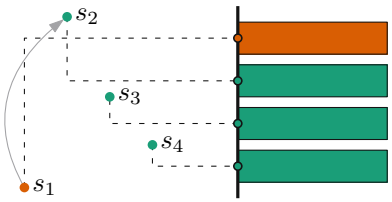
Note that we can retrieve a feasible labeling for $\mathcal{I}$, if it exists, by the same means as for the DP-Algorithm from Chapter 4. Finally, recall that this algorithm only checks whether there exists a feasible labeling. It is not guaranteed that the reported labeling is the best according to an optimization function f.

6.3 Soft Grouping and Ordering Constraints

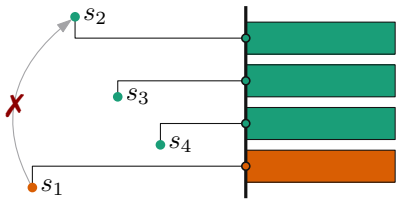
The third and last variation we consider in this thesis is motivated by the fact that grouping and ordering constraints might cause long leaders or prevent any feasible labeling. Consider, for example, the instance from Figure 6.10 that contains a grouping and an ordering constraint. The ordering constraint $s_1 \prec s_2$ requires us to label s_1 above s_2. However, the geometric position of s_2, i.e., above all ports, makes a feasible labeling impossible, as Figure 6.10a shows. Once we ignore this ordering constraint, as in Figure 6.10b, we can obtain a feasible labeling.

Therefore, it could be beneficial to not respect some constraints but pay a penalty instead to obtain an overall more appealing labeling, i.e., consider our semantic constraints as soft constraints. It would be natural to incorporate such a mechanic into our DP-Algorithm from Chapter 4. However, if the given constraints are not respectable, there might exist no PQ-A-Graph $\mathcal{T}$ that represents them, as there might not even exist a PQ-Tree that represents the (conflicting) grouping constraints.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) An instance that does not possess a feasible labeling.



(b) A planar labeling where we do not respect the constraint $s_1 \prec s_2$.

Figure 6.10: An instance that only possesses a feasible labeling if we do not respect the ordering constraint $s_1 \prec s_2$, indicated by the gray arrow.

We present in the following a simple way of working around this limitation by deciding beforehand which constraints we want to ignore. Since we cannot know in advance which of the constraints we should neglect, we try out all possible combinations, yielding an exponential running time. By closer examining the running time, we will see that this problem is fixed parameter tractable (FPT) in the number of constraints that should be respected. Such a result should be preceded by an accompanying NP-hardness proof. Although we suspect the boundary labeling problem in the presence of soft constraints to be NP-hard, finding the corresponding reduction is an open problem.

Let $g : \mathcal{C} \to \mathbb{R}_0^+$ be a function that assigns to each grouping and ordering constraint a non-negative penalty value for not respecting it. For a labeling $\mathcal{L}$, we define $g(\mathcal{L})$ as the sum of penalties for all constraints not respected in $\mathcal{L}$.

Theorem 6.2. *Let $\mathcal{I} = (\mathcal{S}, \mathcal{P}, h, f)$ be an instance of the* 1-Sided Boundary Labeling *problem, with $n = |\mathcal{S}|$ and $m = |\mathcal{P}|$. Furthermore, let $\mathcal{C} = (\mathcal{G}, \prec)$ be k grouping and r ordering constraints, and $g : \mathcal{C} \to \mathbb{R}_0^+$ be a penalty function for not respecting the constraints. We can compute a feasible labeling $\mathcal{L}^*$ of $\mathcal{I}$ that minimizes $f(\mathcal{L}^*) + g(\mathcal{L}^*)$ in $\mathcal{O}(2^{k+r}\,(n^5m^3 \log m + k + \|\mathcal{G}\|) + nmf(\cdot,\cdot))$ time using $\mathcal{O}(n^2m^2)$ space, or conclude that no such labeling exists, where $\mathcal{O}(f(\cdot,\cdot))$ denotes the evaluation time of $f(s,p)$, for arbitrary $s \in \mathcal{S}$ and $p \in \mathcal{P}$, i.e., the evaluation complexity of the optimization function f.*

Proof. For an instance $\mathcal{I}$ and grouping and ordering constraints $\mathcal{C}$, we perform as follows. We enumerate all subsets of possible constraints, i.e., we enumerate all entries of $2^{(\mathcal{G} \cup \prec)}$. For each set of constraints $\mathcal{C}' = (\mathcal{G}', \prec')$ of size k' and r', respectively, we enrich $\mathcal{I}$ by the constraints, thus creating an instance $\mathcal{I}'$ of 1-Sided Constrained Boundary Labeling, and run our DP-Algorithm from Chapter 4 on $\mathcal{I}'$. As shown in Theorem 4.1, this takes $\mathcal{O}(n^5m^3 \log m + k' + \|\mathcal{G}'\| + nmf(\cdot,\cdot))$ time and requires $\mathcal{O}(n^2m^2)$ space. If $\mathcal{I}'$ does not possess a feasible labeling (that respects the constraints $\mathcal{C}'$), we store $E[\mathcal{C}'] = \infty$ in a table E. Otherwise, if $\mathcal{L}$ is the labeling that minimizes f in the presence of the constraints $\mathcal{C}'$, we store $E[\mathcal{C}'] = f(\mathcal{L}) + g(\mathcal{L})$. Furthermore, we maintain in a second table E' for each subset of constraints $\mathcal{C}'$ the optimal labeling. In the end, we report the labeling $\mathcal{L}$ that yielded the minimum but finite entry in E as the labeling $\mathcal{L}^*$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Regarding the running time, we first note that we have $\left|2^{(\mathscr{G}\cup\prec)}\right| = 2^{k+r}$ and $\mathcal{C}' \subseteq \mathcal{C}$. Since the values for f are independent of the constraints, we can compute them once in the beginning. Therefore, using Theorem 4.1, we have a running time of $\mathcal{O}(2^{k+r}\,(n^5m^3\log m + k + \|\mathscr{G}\|) + nmf(\cdot,\cdot))$. Regarding space consumption, we note that we iteratively evaluate one subset $\mathcal{C}'$ of constraints at a time. Therefore, we can re-use the same DP-Tables when iteratively calling the DP-Algorithm from Chapter 4. Furthermore, this means that we do not have to store the whole tables E and E', but only those values corresponding to the so far optimal labeling. Since we can interpret a labeling as a function from the sites to (the indices or y-coordinates of) the ports, we have $\mathcal{O}(|\mathcal{L}|) = \mathcal{O}(n)$. The space requirement is, therefore, dominated by the DP-Tables of our algorithm from Chapter 4, which require $\mathcal{O}(n^2m^2)$ space, as shown in Theorem 4.1. □

Recall from Theorem 4.1 that the terms k and $\|\mathscr{G}\|$ in the running time of our DP-Algorithm arose from the preprocessing steps to obtain the PQ-A-Graph $\mathcal{T}$. As this only depends on the constraints $\mathcal{C}$ (or rather $\mathcal{C}'$), we can summarize these terms in a function h that only depends on (the size of) $\mathcal{C}$ and obtain the following corollary.

Corollary 6.4. 1-Sided Boundary Labeling *for n sites $\mathcal{S}$, m ports $\mathcal{P}$, soft constraints $\mathcal{C} = (\mathscr{G}, \prec)$, and a penalty function g, can be solved in $\mathcal{O}(h(\mathcal{C})n^5m^3\log m)$ time and $\mathcal{O}(n^2m^2)$ space for a computable function h. Thus, it is FPT in the size of the soft constraints.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Conclusion

This thesis aimed to make a first step towards supporting semantic constraints in external labeling by extending the common boundary labeling model by grouping and ordering constraints. We have seen that most of the existing work on external labeling does not take such constraints into account. The papers that do support them all have some limitations: They either make compromises by allowing leaders to cross or grouping constraints to be violated or see it as a possible extension without analyzing the impact that such constraints can have on the running time. Therefore, we strived for approaches tailored to these constraints and started by formally defining grouping and ordering constraints for one-sided boundary labelings: When arranging the sites according to the order of their labels on the right boundary, the sites of each group must appear consecutively, and the resulting total order must be an extension of the ordering constraints. For multi-sided boundary labelings, we require that we must not split groups across different sides of the boundary, and ordering constraints are only relevant if both sites are on the same side of the boundary.

Using these definitions, we managed to adapt existing results from the literature to the extended model. On the one hand, we showed that finding a one-sided labeling with non-uniform height labels is weakly NP-complete for sliding reference points, even with only a constant number of grouping or ordering constraints. On the other hand, we proposed a polynomial-time dynamic programming algorithm to compute a feasible one-sided boundary labeling with *po*-leaders, uniform height labels, and fixed reference points that respects a set of constraints $\mathcal{C}$ and minimizes a function $f : \mathcal{S} \times \mathcal{P} \to \mathbb{R}_0^+$ or report that no feasible labeling exists. Our algorithm operates on PQ-A-Graphs, an extension of PQ-Trees we introduced, that are capable of efficiently encoding all respectable constraints. Surprisingly, in contrast to the unconstrained boundary labeling problem, our results do not carry over to multi-sided settings, and we showed that already finding a feasible labeling for two sides is NP-complete.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Furthermore, we implemented our algorithm and evaluated its performance experimentally. The experiments revealed that blindly adding semantic constraints, especially ordering constraints, to instances tends to turn them infeasible. Furthermore, although we can compute labelings for our motivating example, i.e., medical drawings, within four seconds, computing them for larger instances, such as cartographic maps, can take up to seven minutes. These results motivated us to look into variations of our problem: Sliding reference points, permitting only pairwise disjoint grouping constraints and no ordering constraints, and allowing restrictions to be violated at the cost of a penalty. While we consider the first variation mostly solved, we see the latter two as candidates for complementing our results in future work.

But not only the variations are directions for future research. While we have answered in this thesis some open questions related to semantic constraints in external labeling, we have stumbled upon others. We see the high running time of our algorithm as a limitation of our approach. Therefore, reducing it or finding heuristic approaches or approximation algorithms is a major open challenge. For example, we might apply other algorithmic paradigms to find at least some feasible labeling fast. Although there are no theoretical guarantees, the experiments have shown that state-of-the-art ILP solvers are powerful enough to find optimal solutions for many instances in a (comparatively) short time. Hence, we should try to formulate b-Sided Constrained Boundary Labeling as an ILP and experimentally evaluate its running time. The experiments revealed that our constraints, especially ordering constraints, tend to turn an instance infeasible. Therefore, we deem it crucial to be able to interpret grouping and ordering constraints not only as hard but also as soft constraints. The computational complexity of the corresponding problem is still open, as well as the one of 2-Sided Constrained Boundary Labeling once we consider only grouping or ordering constraints, but not in combination. While we have mainly compared the leader length of labelings with constraints to their counterparts without constraints, a user study is worth conducting to analyze the impact of complying with semantic constraints from the perspective of a user in more detail. Finally, although grouping and ordering constraints are arguably the most common manifestation of semantic constraints, they are not the only ones, and we see identifying and supporting other (semantic) constraints as an interesting open problem in its own right.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[Alg23] Algorithms and Complexity Group. *Compute Cluster*. Accessed on 2023-10-14. 2023. URL: https://www.ac.tuwien.ac.at/students/compute-cluster/.

[Ang23] Angular. *Angular: The web development framework for building the future*. Accessed on 2023-08-27. 2023. URL: https://angular.io/.

[BČČ23] Petr Bobák, Ladislav Čmolík, and Martin Čadík. "Reinforced Labels: Multi-Agent Deep Reinforcement Learning for Point-Feature Label Placement". In: *IEEE Transactions on Visualization and Computer Graphics* (2023). To appear, pp. 1–14. DOI: 10.1109/tvcg.2023.3313729.

[BCF+15] Michael A. Bekos, Sabine Cornelsen, Martin Fink, Seok-Hee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis. "Many-to-One Boundary Labeling with Backbones". In: *Journal of Graph Algorithms and Applications (JGAA)* 19.3 (2015), pp. 779–816. DOI: 10.7155/jgaa.00379.

[BCK+18] Prosenjit Bose, Paz Carmi, J. Mark Keil, Saeed Mehrabi, and Debajyoti Mondal. "Boundary Labeling for Rectangular Diagrams". In: *Proc. 16th Scandinavian Workshop on Algorithm Theory (SWAT)*. Vol. 101. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 12:1–12:14. DOI: 10.4230/LIPICS.SWAT.2018.12.

[BCKO08] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008.

[BDW98] Rainer E. Burkard, Vladimir G. Deineko, and Gerhard J. Woeginger. "The Travelling Salesman and the PQ-Tree". In: *Mathematics of Operations Research* 23.3 (1998), pp. 613–623. DOI: 10.1287/moor.23.3.613.

[BF00] Michael A. Bender and Martin Farach-Colton. "The LCA Problem Revisited". In: *Proc. 4th Latin American Symposium on Theoretical Informatics (LATIN)*. Vol. 1776. Lecture Notes in Computer Science (LNCS). Springer, 2000, pp. 88–94. DOI: 10.1007/10719839_9.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[BGNN19] Lukas Barth, Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. "On the readability of leaders in boundary labeling". In: *Information Visualization* 18.1 (2019), pp. 110–132. DOI: 10.1177/1473871618799500.

[BHKN09] Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. "Algorithms for Multi-Criteria Boundary Labeling". In: *Journal of Graph Algorithms and Applications (JGAA)* 13.3 (2009), pp. 289–317. DOI: 10.7155/jgaa.00189.

[BKNS10] Michael A. Bekos, Michael Kaufmann, Martin Nöllenburg, and Antonios Symvonis. "Boundary Labeling with Octilinear Leaders". In: *Algorithmica* 57.3 (2010), pp. 436–461. DOI: 10.1007/s00453-009-9283-6.

[BKPS10] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. "Area-Feature Boundary Labeling". In: *The Computer Journal* 53.6 (2010), pp. 827–841. DOI: 10.1093/comjnl/bxp087.

[BKPS11] Michael A. Bekos, Michael Kaufmann, Dimitrios Papadopoulos, and Antonios Symvonis. "Combining Traditional Map Labeling with Boundary Labeling". In: *Proc. 37th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*. Vol. 6543. Lecture Notes in Computer Science (LNCS). Springer, 2011, pp. 111–122. DOI: 10.1007/978-3-642-18381-2_9.

[BKSW07] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. "Boundary labeling: Models and efficient algorithms for rectangular maps". In: *Computational Geometry* 36.3 (2007), pp. 215–236. DOI: 10.1016/j.comgeo.2006.05.003.

[BL76] Kellogg S. Booth and George S. Lueker. "Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms". In: *Journal of Computer and System Sciences (JCSS)* 13.3 (1976), pp. 335–379. DOI: 10.1016/S0022-0000(76)80045-1.

[BMs23] Satyam Bhuyan, Santanu Mukherjee, and (sciencefacts.net). *Layers of the Sun*. Accessed on 2023-09-07. 2023. URL: https://www.sciencefacts.net/layers-of-the-sun.html.

[BNN21] Michael A. Bekos, Benjamin Niedermann, and Martin Nöllenburg. *External Labeling: Fundamental Concepts and Algorithmic Techniques*. Synthesis Lectures on Visualization. Springer, 2021. DOI: 10.1007/978-3-031-02609-6.

[BOH11] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. "D^3 Data-Driven Documents". In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2301–2309. DOI: 10.1109/TVCG.2011.185.

[Bri90] Mary Helen Briscoe. *A Researcher's Guide to Scientific and Medical Illustrations*. Springer Science & Business Media, 1990. DOI: 10.1007/978-1-4684-0355-8.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. DOI: 10.1007/978-3-319-21275-3.

[CNW83] Gérard Cornuéjols, George Nemhauser, and Laurence Wolsey. *The Uncapacitated Facility Location Problem*. Tech. rep. Cornell University Operations Research and Industrial Engineering, 1983.

[CPWN22] Ladislav Cmolík, Vaclav Pavlovec, Hsiang-Yun Wu, and Martin Nöllenburg. "Mixed Labeling: Integrating Internal and External Labels". In: *IEEE Transactions on Visualization and Computer Graphics* 28.4 (2022), pp. 1848–1861. DOI: 10.1109/TVCG.2020.3027368.

[DDG+11] Erik Dahlström, Patrick Dengler, Anthony Grasso, Chris Lilley, Cameron McCormack, Doug Schepers, Jonathan Watt, Jon Ferraiolo, Jun Fujisawa, and Dean Jackson. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. Accessed on 2023-08-27. 2011. URL: https://www.w3.org/TR/2011/REC-SVG11-20110816.

[FG65] Delbert Fulkerson and Oliver Gross. "Incidence matrices and interval graphs". In: *Pacific Journal of Mathematics* 15.3 (1965), pp. 835–855.

[FHS+12] Martin Fink, Jan-Henrik Haunert, André Schulz, Joachim Spoerhase, and Alexander Wolff. "Algorithms for Labeling Focus Regions". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2583–2592. DOI: 10.1109/TVCG.2012.193.

[FL05] Christodoulos A. Floudas and Xiaoxia Lin. "Mixed Integer Linear Programming in Process Scheduling: Modeling, Algorithms, and Applications". In: *Annals of Operations Research* 139.1 (2005), pp. 131–162. DOI: 10.1007/s10479-005-3446-x.

[FP99] Jean-Daniel Fekete and Catherine Plaisant. "Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization". In: *Proc. 17th ACM Conference on Human Factors in Computing Systems (CHI)*. CHI '99. Association for Computing Machinery (ACM), 1999, pp. 512–519. DOI: 10.1145/302979.303148.

[FPR21] Simon D. Fink, Matthias Pfretzschner, and Ignaz Rutter. "Experimental Comparison of PC-Trees and PQ-Trees". In: *Proc. 29th European Symposium on Algorithms (ESA)*. Vol. 204. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 43:1–43:13. DOI: 10.4230/LIPIcs.ESA.2021.43.

[FS16] Martin Fink and Subhash Suri. "Boundary Labeling with Obstacles". In: *Proc. 28th Canadian Conference on Computational Geometry (CCCG)*. Simon Fraser University, 2016, pp. 86–92.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[FW91] Michael Formann and Frank Wagner. “A Packing Problem with Applications to Lettering of Maps”. In: *Proc. 7th International Symposium on Computational Geometry (SoCG)*. SCG ’91. Association for Computing Machinery (ACM), 1991, pp. 281–288. DOI: 10.1145/109648.109680.

[GAH23] Sven Gedicke, Lukas Arzoumanidis, and Jan-Henrik Haunert. “Automating the external placement of symbols for point features in situation maps for emergency response”. In: *Cartography and Geographic Information Science (CaGIS)* 50.4 (2023), pp. 385–402. DOI: 10.1080/15230406.2023.2213446.

[GBNH21] Sven Gedicke, Annika Bonerath, Benjamin Niedermann, and Jan-Henrik Haunert. “Zoomless Maps: External Labeling Methods for the Interactive Exploration of Dense Point Sets at a Fixed Map Scale”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), pp. 1247–1256. DOI: 10.1109/TVCG.2020.3030399.

[GHN15] Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. “Multirow Boundary-Labeling Algorithms for Panorama Images”. In: *ACM Transactions on Spatial Algorithms and Systems (TSAS)* 1.1 (2015), 1:1–1:30. DOI: 10.1145/2794299.

[GHS06a] Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. “Agent-Based Annotation of Interactive 3D Visualizations”. In: *Proc. 6th International Symposium on Smart Graphics (SG)*. Vol. 4073. Lecture Notes in Computer Science (LNCS). Springer, 2006, pp. 24–35. DOI: 10.1007/11795018_3.

[GHS06b] Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. “Contextual Grouping of Labels”. In: *Proc. 17th Simulation und Visualisierung (SimVis)*. SCS Publishing House e.V., 2006, pp. 245–258.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Gru93] Thomas R. Gruber. “A translation approach to portable ontology specifications”. In: *Knowledge Acquisition* 5.2 (1993), pp. 199–220. DOI: 10.1006/knac.1993.1008.

[Gur23a] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. Accessed on 2023-08-27. 2023. URL: https://www.gurobi.com.

[Gur23b] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual: GRBModel::write()*. Accessed on 2023-08-27. 2023. URL: https://www.gurobi.com/documentation/current/refman/cpp_model_write.html.

[Gur23c] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual: Logging*. Accessed on 2023-08-27. 2023. URL: https://www.gurobi.com/documentation/current/refman/logfile.html.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[HGAS05] Knut Hartmann, Timo Götzelmann, Kamran Ali, and Thomas Strothotte. "Metrics for Functional and Aesthetic Label Layouts". In: *Proc. 5th International Symposium on Smart Graphics (SG)*. Vol. 3638. Lecture Notes in Computer Science (LNCS). Springer, 2005, pp. 115–126. DOI: 10.1007/11536482_10.

[HM03] Wen-Lian Hsu and Ross M. McConnell. "PC trees and circular-ones arrangements". In: *Theoretical Computer Science* 296.1 (2003), pp. 99–116. DOI: 10.1016/S0304-3975(02)00435-8.

[HPL14] Zhi-Dong Huang, Sheung-Hung Poon, and Chun-Cheng Lin. "Boundary Labeling with Flexible Label Positions". In: *Proc. 8th International Conference and Workshops on Algorithms and Computation (WALCOM)*. Vol. 8344. Lecture Notes in Computer Science (LNCS). Springer, 2014, pp. 44–55. DOI: 10.1007/978-3-319-04657-0_7.

[Hsu01] Wen-Lian Hsu. "PC-Trees vs. PQ-Trees". In: *Proc. 7th International Computing and Combinatorics Conference (COCOON)*. Vol. 2108. Lecture Notes in Computer Science (LNCS). Springer, 2001, pp. 207–217. DOI: 10.1007/3-540-44679-6_23.

[Imh75] Eduard Imhof. "Positioning names on maps". In: *The American Cartographer* 2.2 (1975), pp. 128–144.

[JLCZ20] Haitao Jiang, Hong Liu, Cédric Chauve, and Binhai Zhu. "Breakpoint distance and PQ-trees". In: *Information and Computation* 275 (2020), p. 104584. DOI: 10.1016/j.ic.2020.104584.

[JSV98] Brigitte Jaumard, Frédéric Semet, and Tsevi Vovor. "A generalized linear programming model for nurse scheduling". In: *European Journal of Operational Research* 107.1 (1998), pp. 1–18. DOI: 10.1016/S0377-2217(97)00330-5.

[Kau09] Michael Kaufmann. "On Map Labeling with Leaders". In: *Efficient Algorithms, Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*. Vol. 5760. Lecture Notes in Computer Science (LNCS). Springer, 2009, pp. 290–304. DOI: 10.1007/978-3-642-03456-5_20.

[Kir19] Melanie Kirchgessner. *Waldlehrpfad Fürth*. (www.artofdesign-online.de), Accessed on 2023-09-07. 2019. URL: https://artofdesign-online.de/stadtwald.

[KKO+17] Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, and Tomás Vyskocil. "Extending Partial Representations of Interval Graphs". In: *Algorithmica* 78.3 (2017), pp. 945–967. DOI: 10.1007/s00453-016-0186-z.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[KKS+23] Jonathan Klawitter, Felix Klesen, Joris Y. Scholl, Thomas C. van Dijk, and Alexander Zaft. “Visualizing Geophylogenies - Internal and External Labeling with Phylogenetic Tree Constraints”. In: *Proc. 12th International Conference Geographic Information Science (GIScience)*. Vol. 277. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 5:1–5:16. DOI: 10.4230/LIPIcs.GIScience.2023.5.

[KLW14] Philipp Kindermann, Fabian Lipp, and Alexander Wolff. “Luatodonotes: Boundary Labeling for Annotations in Texts”. In: *Proc. 22nd International Symposium Graph Drawing and Network Visualization (GD)*. Vol. 8871. Lecture Notes in Computer Science (LNCS). Springer, 2014, pp. 76–88. DOI: 10.1007/978-3-662-45803-7_7.

[KNR+16] Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff. “Multi-sided Boundary Labeling”. In: *Algorithmica* 76.1 (2016), pp. 225–258. DOI: 10.1007/s00453-015-0028-4.

[LNS16] Maarten Löffler, Martin Nöllenburg, and Frank Staals. “Mixed Map Labeling”. In: *Journal of Spatial Information Science (JOSIS)* 13.1 (2016), pp. 3–32. DOI: 10.5311/JOSIS.2016.13.264.

[LPT+11] Chun-Cheng Lin, Sheung-Hung Poon, Shigeo Takahashi, Hsiang-Yun Wu, and Hsu-Chun Yen. “One-and-a-Half-Side Boundary Labeling”. In: *Proc. 5th International Conference on Combinatorial Optimization and Applications (COCOA)*. Vol. 6831. Lecture Notes in Computer Science (LNCS). Springer, 2011, pp. 387–398. DOI: 10.1007/978-3-642-22616-8_30.

[LPW05] Gad M. Landau, Laxmi Parida, and Oren Weimann. “Using PQ Trees for Comparative Genomics”. In: *Proc. 16th Annual Symposium on Combinatorial Pattern Matching (CPM)*. Vol. 3537. Lecture Notes in Computer Science (LNCS). Springer, 2005, pp. 128–143. DOI: 10.1007/11496656_12.

[LRT21] Giuseppe Liotta, Ignaz Rutter, and Alessandra Tappini. “Simultaneous FPQ-ordering and hybrid planarity testing”. In: *Theoretical Computer Science* 874 (2021), pp. 59–79. DOI: 10.1016/j.tcs.2021.05.012.

[May89] Richard E. Mayer. “Systematic thinking fostered by illustrations in scientific text.” In: *Journal of Educational Psychology* 81.2 (1989), pp. 240–246. DOI: 10.1037/0022-0663.81.2.240.

[MG90] Richard E. Mayer and Joan K. Gallini. “When is an illustration worth ten thousand words?” In: *Journal of Educational Psychology* 82.4 (1990), pp. 715–726. DOI: 10.1037/0022-0663.82.4.715.

[MPT98] Joao Meidanis, Oscar Porto, and Guilherme P. Telles. “On the Consecutive Ones Property”. In: *Discrete Applied Mathematics* 88.1-3 (1998), pp. 325–354. DOI: 10.1016/S0166-218X(98)00078-X.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Net81] Frank H. M. D. Netter. “Graphics in Medical Illustration”. In: *Neurosurgery* 28 (1981), pp. 225–232. DOI: 10.1093/neurosurgery/28.cn_suppl_1.225.

[NNR17] Benjamin Niedermann, Martin Nöllenburg, and Ignaz Rutter. “Radial Contour Labeling with Straight Leaders”. In: *Proc. 10th IEEE Pacific Visualization Symposium (PacificVis)*. PacificVis ’17. IEEE Computer Society, 2017, pp. 295–304. DOI: 10.1109/PACIFICVIS.2017.8031608.

[Nor23a] NordNordWest (Wikimedia Commons). *File:Austria adm location map.svg*. Licence: CC BY-SA 3.0 (DE), Accessed on 2023-10-29. 2023. URL: https://commons.wikimedia.org/wiki/File:Austria_adm_location_map.svg.

[Nor23b] NordNordWest (Wikimedia Commons). *File:Germany adm location map.svg*. Licence: CC BY-SA 3.0 (DE), Accessed on 2023-10-29. 2023. URL: https://commons.wikimedia.org/wiki/File:Germany_adm_location_map.svg.

[Nor23c] NordNordWest (Wikimedia Commons). *File:Italy location map.svg*. Licence: CC BY-SA 3.0 (DEED), Accessed on 2023-10-29. 2023. URL: https://commons.wikimedia.org/wiki/File:Italy_location_map.svg.

[NPS10] Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. “Dynamic one-sided boundary labeling”. In: *Proc. 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS)*. GIS ’10. Association for Computing Machinery (ACM), 2010, pp. 310–319. DOI: 10.1145/1869790.1869834.

[Sch78] Thomas J. Schaefer. “The Complexity of Satisfiability Problems”. In: *Proc. 10th Symposium on the Theory of Computing (STOC)*. STOC ’78. Association for Computing Machinery (ACM), 1978, pp. 216–226. DOI: 10.1145/800133.804350.

[SH99] Wei-Kuan Shih and Wen-Lian Hsu. “A New Planarity Test”. In: *Theoretical Computer Science* 223.1-2 (1999), pp. 179–191. DOI: 10.1016/S0304-3975(98)00120-0.

[Sim23] SimpleMaps.com. *simplemaps: Interactive Maps & Data*. Accessed on 2023-10-13. 2023. URL: https://simplemaps.com/.

[Spa06] Werner Spalteholz. *Hand Atlas of Human Anatomy*. 2nd Edition. Edited and translated from the fourth German edition by Lewellys F. Barker, Licence: CC BY-NC 4.0. JB Lippincott, 1906.

[Tuf01] Edward R. Tufte. *The visual display of quantitative information*. 2nd Edition. Graphics Press, 2001.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[VVAH07] Ian Vollick, Daniel Vogel, Maneesh Agrawala, and Aaron Hertzmann. "Specifying label layout style by example". In: *Proc. 20th ACM Symposium on User Interface Software and Technology (UIST)*. UIST '07. Association for Computing Machinery (ACM), 2007, pp. 221–230. DOI: 10.1145/1294211.1294252.

[WDM18] Luca Weihs, Mathias Drton, and Nicolai Meinshausen. "Symmetric rank covariances: a generalized framework for nonparametric measures of dependence". In: *Biometrika* 105.3 (2018), pp. 547–562. DOI: 10.1093/biomet/asy021.

[Wei20] Luca (Lucaweihs) Weihs. *GitHub: C++ Range Tree Data Structure*. Accessed on 2023-10-29. 2020. URL: https://github.com/Lucaweihs/range-tree.

[Wol20] Laurence A. Wolsey. *Integer Programming*. 2nd Edition. John Wiley & Sons, Ltd, 2020. DOI: 10.1002/9781119606475.

[Wol99] Alexander Wolff. "Automated Label Placement in Theory and Practice". PhD thesis. Freie Universität Berlin, 1999. DOI: 10.17169/refubium-7108.

[WP13] Jens Waschke and Friedrich Paulsen. *Sobotta Atlas of Human Anatomy: Head, Neck and Neuroanatomy*. 15th Edition. Vol. 3. Elsevier, 2013.

[Yoe72] Pinhas Yoeli. "The Logic of Automated Map Lettering". In: *The Cartographic Journal* 9.2 (1972), pp. 99–108. DOI: 10.1179/caj.1972.9.2.99.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN