FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

An Interactive Optimization Framework for Point Feature Label Placement

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Raphael Löffler, B.Sc.
Registration Number 1025967

to the Faculty of Informatics

at the TU Wien

Advisor: Assoc.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Assistance: Univ. Ass. Fabian Klute, M.Sc., B.Sc

Vienna, 9th August, 2018

Raphael Löffler | Martin Nöllenburg

Erklärung zur Verfassung der Arbeit

Raphael Löffler, B.Sc.

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, 9. August 2018

Raphael Löffler

Abstract

This thesis describes an interactive user-centered and web-based optimization framework for the point feature label placement problem. It tries to fill the gap between the time consuming and expensive manual map creation and the less qualitative but much faster and cheaper automatic label placement to eradicate the disadvantages of both approaches. Given a set of point features we want to place for each feature a label on the map such that the labels do not overlap. Since a high-quality map has to furthermore satisfy cartographic guidelines and preferences, our developed framework allows an expert user to add his or her domain knowledge in an interactive way by improving and updating an initial labeling step by step. We therefore collected and formally defined user constraints through interviews with a cartographic expert.

In this work we consider axis-aligned rectangular labels with predetermined fixed label candidate positions in a fixed four position model on static geographical maps. We describe the point feature label placement problem with a conflict graph and use it as the basis for our algorithms. We present and implement in this thesis both existing and new algorithms for the automatic point feature label placement. This includes simple algorithms, exact algorithms and several heuristics for the label number maximization problem as well as the minimum number of conflicts problem.

Finally, we investigate and evaluate our framework and its algorithms regarding the performance and the quality of the created labeling solutions on real world data. We hereby also concentrate on identifying well performing algorithm combinations for various kind of data sets as they are used in typical use cases of our framework.

Kurzfassung

Diese Diplomarbeit beschreibt ein interaktives Benutzer-zentriertes und Web-basiertes Optimierungsframework für das Point Feature Label Placement Problem. Die Arbeit versucht die Lücke zwischen der zeitaufwändigen und teuren manuellen Kartenerstellung und der weniger qualitativen aber wesentlich schnelleren und billigeren automatischen Label Platzierung zu schließen und die Nachteile beider Ansätze auszumerzen. Für ein gegebenes Set an punktuellen Features wollen wir für jedes Feature ein Label auf der Karte so platzieren, dass sich keine Labels überlappen. Da eine Karte mit hoher Qualität zusätzlich noch verschiedene kartographische Richtlinien und Präferenzen erfüllen muss, erlaubt unser entwickeltes Framework einem Expertenbenutzer sein bereichsspezifisches Fachwissen in einem interaktiven Vorgang einzubringen, um so eine initiale Kartenbeschriftung schrittweise zu verbessern. Wir haben dafür durch Interviews mit einer Kartographin diese benutzerdefinierten Bedingungen und Beschränkungen gesammelt und formal definiert.

In dieser Arbeit betrachten wir nur achsenparallele, rechteckige Labels mit vordefinierten fixen Positionen in einem fixen Vier-Positionen-Model in statischen geographischen Karten. Wir haben das Point Feature Label Placement Problem mittels eines Konfliktgraphen beschrieben und verwenden diesen als Basis für all unsere Algorithmen. Wir haben in dieser Arbeit sowohl existierende als auch neue Algorithmen für automatisches Point Feature Label Placement präsentiert und implementiert. Dies beinhaltet einfache Algorithmen, exakte Algorithmen und verschiedene Heuristiken für das Problem der Labelmaximierung und dem Problem der Konfliktminimierung.

Letztlich untersuchen und evaluieren wir unser Framework und seine Algorithmen bezüglich der Performance und der Qualität der erzeugten Lösungen mit realen Daten. Wir konzentrieren uns hier auch auf performante Algorithmenkombinationen für verschiedenartige Datensätze wie sie im normalen Gebrauch des Frameworks verwendet werden.

Contents

CHAPTER 1

Introduction

Label placement is an important task in information visualization and especially in the area of cartography where it is known as the map-labeling problem. Points of interest of different spatial characteristics (i.e., point, line or area features) shall be described and identified for example by a text element or an icon (a so called *label*) that is attached to the feature. When creating (geographic) maps or diagrams each feature shall be labeled so that no two labels intersect or overlap each other or obscure another feature. Additionally cartographic placement guidelines and preferences (e.g., location, orientation, shape, size) have to be obeyed. The quality of a map in the sense of legibility and clarity is strongly dependent on the position of the labels for the features.

As the cartographic label placement is a very time-consuming task, large efforts have been made to automatize the process of label placement. Different objectives like maximizing the number of labeled features or maximizing the size of the labels can be pursued. These are geometric optimization problems and are well known in the literature. The problem of finding an optimal solution for both problems is known to be NP-hard [WWKS01, FW91].

For the label placement problem several exact approaches, practically effective heuristics and approximations have been developed. But yet, after years of research, maps created by automatic label placement algorithms do not meet the quality of those created by human experts in cartography. Hence, in order not to release sub-optimal maps, the automatically created maps need to be manually post-processed by a domain expert, which is a very tedious procedure, or as an alternative, do a solely manual creation of maps, which is even more time consuming and expensive.

With this thesis we want to fill the gap between high quality but expensive manual map creation and the cheap and fast but less qualitative automatic map labeling. We investigate an approach where the domain experts work hand in hand with the automatic label placement algorithms during the whole map creation process. The idea is that different algorithms or heuristics produce an initial labeling solution and then the domain

expert improves and refines the labeling step by step by adding different constraints (i.e., his domain knowledge). The algorithms in the background then use these constraints to recalculate an optimal labeling and improve the previous solution. We concentrate on the cartographic point feature label placement problem on static geographical maps with a discrete label positioning model and axis-parallel rectangular labels.

In particular we give in this thesis a formal definition of the problem, collect and formally define the user constraints through interviews with a domain expert and implement, based on these, a prototype of a web-based user-centered map creation tool. This tool is intended to be used by domain experts in cartography to speed up their map creation process and ensure a high quality of the created maps.

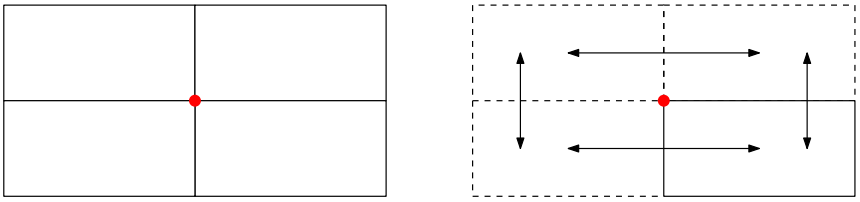
We study the complexity of the used algorithms and perform quality and performance evaluation with real geospatial data from OpenStreetMap [osm]. Since our approach for an automated human-centered label placement has underlying dynamic geometric and combinatorial optimization problems, the outcome is also of interest for the algorithmic community in general and may be applied to other problems too.

1.1 Related Work

Good label placement has been a significant challenge throughout cartographic history. The placement of textual names for various types of features on a cartographic map is highly significant to the quality of the resulting map [Imh75]. Since the manual map labeling process can take up to more than 50 percent of the map production time [Yoe72], the need of automation and use of computers for placing labels was seen already back in the 1970s, e.g., by Yoeli [Yoe72]. Ever since then, there were many efforts made to find approaches to address the automatization of label placement. It turned out quickly, that the problem of label placement, regardless of the features being labeled, leads to combinatorial optimization problems that are NP-hard for most of its practical variants [MS91, CMS95]. Exact algorithms are only able to solve problem instances with a few hundred of features. Consequently the researchers focused on finding approximations and practically effective heuristics for objectives like the number of labels maximization or label size maximization. A good list of publications surveying the label placement problem until the year 2009 can be seen in "the map labeling bibliography" [map].

Most works in the literature for automated label placement concentrate on *static* maps. The primary goal for static maps is to maximize the information content on the map like maximizing the number of labels placed. *Dynamic* maps on the contrary are intended for navigation. They are characterized by allowing the user to zoom (change the scale), pan (change the region of interest) and rotate the map view and hence the labels have to be placed in interactive speed. Since label placement on static maps is NP-hard, algorithms and solutions for static maps are inadequate for dynamic maps.

Been et al. [BDY06] were one of the first that investigated label placement on dynamic maps. They focused only on zooming and panning and identified that for dynamic maps



(a) (b)

Figure 1.1: The (a) discrete model with four fixed positions for the label candidates, and the (b) slider model where the label candidates can slide around the point feature.

distracting behaviour such as label popping (i.e., labels that disappear and reappear) or labels that move about (i.e., labels that change their position or size in an unexpected way) during monotonous map navigation shall be avoided.

In further work Been et al. [BNPW10] developed a model for consistent dynamic labeling where each label is represented by a 3d-solid with scale as third dimension. Label popping is avoided during zooming by calculating the so-called *active range* (i.e., a single range of scales where the label is selected) for each label, so that no two selected labels intersect at any scale. Maximizing the sum of heights of these ranges is then equivalent to the goal of maximizing the number of labels selected at each scale.

Gemsa et al. [GNR11] did some research on dynamic rotating maps. They take a static labeled map as input and try to find a consistent labeling of non-overlapping and non-obscuring labels while rotating the map. The number of visible labels for all rotation angles shall be maximal.

There exist many different position models for the label candidates of a feature in the literature. While in cartographic maps the most common approach is to use *internal* labels (labels placed in the immediate neighbourhood to the feature), in other application areas like medicine and engineering *external* labels (labels placed outside the picture and connected to the feature with a leader line) are used more frequently (e.g., for anatomic or technical illustrations and drawings) [BKSW07]. In the literature external labeling is also know as *boundary labeling* and has the advantage to manage labeling on very dense feature areas. The drawback of boundary labeling is that it is often more difficult to see to which feature a label belongs [LNS16]. Löffler et al. [LNS16] presented an approach for mixed map labeling where each feature has to be labeled with either an internal or an external label while maximizing the internal labels.

For internal labeling a popular approach is to create several candidate label positions and then select one of them according to some labeling preferences and objectives. Basically there are two types of positioning models for internal label placement namely the *discrete model* and the *slider model* [FW91, vKSW98] (see Figure 1.1).

In a discrete positioning model the location and the number of the label candidates are predefined. That means that each feature has a set of explicit enumerated possible label

positions [ArDT09, MRL10]. All candidates are directly adjacent to the feature and are of identical size. During the label placement one of the candidates is selected and assigned to the feature.

The slider positioning model allows the label candidate to slide around the feature, i.e., the label candidate can touch the feature anywhere on its edges or in its corners. Hence, in the slider model there is a continuous space with an infinite number of possible positions for placing the label [MRL10]. The advantage of the slider model is that for the same input in general more labels can be placed without overlaps than in the discrete model.

Much research concentrates on the discrete positioning model for point feature label placement. Christensen et al. [CMS95] proposed methods based on simulated annealing and a discrete form of gradient descent while Ribeiro et al. [RL06, RML11], Mauri et al. [MRL10] and Zoraster [Zor86, Zor90] developed different approaches based on integer linear programming.

Formann and Wagner [FW91] investigated the labeling problem in relation to the SAT problem. They allowed the features to have two possible label candidate positions and encoded the relations of intersecting candidates as clauses of a 2-SAT instance. Jung and Chwa [JC04] took up this approach and extended it by suggesting a new encoding rule where each feature can have four candidates. They present a 2-approximation algorithm for non-uniform rectangle labels.

Since the label placement problem can also be described with a so-called conflict graph, where each edge represents a conflict between two label candidates, the labeling problem can be related to finding a maximum independent set in this graph. Approximation algorithms for this kind of problem are presented by Agarwal et al. [AvKS98], Verweij and Aardal [VA99] or Strijk et al. [SVA00]. More recent approaches try to reduce the input size (i.e., the size of the conflict graph) in a preprocessing step [AKCF$^+$04, GD06, BT07, AI16, Str16].

Wagner et al. [WWKS01] focused on the combinatorial characteristic of the point feature label placement with a discrete positioning model and proposed three rules to also reduce the size of the conflict graph by placing and deleting several label candidates that do not alter an optimal solution. These reduction rules are part of the algorithm and hence are not done as a preprocessing step. This approach does not influence the size of an optimal solution.

Less research was done in the area of line and area feature labeling. Niedermann and Nöllenburg [NN16] as well as Wolff et al. [WKvK$^+$00] proposed approaches for efficient line feature labeling and Rylov and Reimer [RR17] developed an algorithm for area feature labeling.

Even though there is much research done on map labeling and automated label placement, automation is not applied broadly in commercial fields but the map labeling process is often done manually or with little support of automatic tools [dNE08]. The main issue is that many of these automatic labeling approaches do not include the subjective

knowledge of human domain experts such as aesthetic criteria or cognitive aspects of human vision.

One promising approach was presented by do Nascimento and Eades [dNE08]. They developed an interactive framework for point feature labeling where a domain expert user can add his or her domain knowledge to the map creation process by adding so called *hints*. Starting from an initial calculated solution the framework allows to continuously refine the labeling solution by inserting additional constraints, calling optimization tools and visualizing the solution. This loop can be repeated until the labeling result meets the users subjective criteria in the sense of aesthetics and good visual appearance of the map. In this process the user is responsible for detecting overlaps and ambiguities. He has to apply the algorithms on regions with label overlaps or ambiguous regions. Optimization is done with two implemented optimization algorithms based on simulated annealing and hill climbing.

This technique may lead to better labeling solutions than other automatic methods and saves much of the monotonic work required to create a high quality map by hand. However, they do not show any evaluation on the effectiveness of their framework and the optimality of the produced labeling solutions. In this thesis we will implement different and more powerful optimization algorithms, evaluate and compare them and align the implemented user constraints and workflows to the needs of cartographic domain experts.

1.2 Outline

The remainder of this thesis is organized as follows. We first describe in Chapter 2 the general label placement problem and its objectives. We sketch notations and label placement strategies used in this thesis as well as useful concepts in graph theory. In Chapter 3 we analyse and describe data structures suitable for storing our spatial input data and for formally describing and efficiently solving our labeling problem. Chapter 4 then presents different automatic labeling algorithms used in the literature and useful for this thesis. In Chapter 5 we collect and analyse the user constraints from a cartographic expert and formalize them so that we can use them in our algorithms.

Chapter 6 then describes our developed framework. We give an overview about the used technology, the implemented data structures, algorithms and label modifications, and describe the user interface as well as the functionality of the prototype. We evaluate our labeling approach and our prototype in terms of performance and solution quality in Chapter 7. Lastly we summarize our work in Chapter 8 and give further prospects for future research.

CHAPTER 2

Preliminaries

In this chapter we introduce and formulate the general label placement problem. We focus on the point feature label placement and describe notations, position models and different objectives as well as basic graph theory notions and aspects that we use throughout this thesis. All other prerequisites are introduced when needed.

2.1 The Label Placement Problem

The general (cartographic) label placement problem can be informally described as follows: Given are a map and a set of graphical features to be labelled. Basically these features can be points, lines or polygons [RML11] where points represent, e.g, cities or villages, lines represent, e.g., roads or rivers and polygons represent area features, e.g., countries or districts. The task in label placement is to assign for each feature a label (e.g., a text or an icon) so that the feature is uniquely identified. See Figure 2.1 for an example labeling of places in Iceland. Most often it is also required that the labels do not intersect or overlap each other or obscure another feature. Additionally, cartographic placement guidelines and preferences (e.g., location, orientation, shape, size) have to be obeyed. The quality of a map in the sense of legibility and clarity is strongly dependent on the position of the label for a feature.

We can formulate three important requirements for good label placement similar to Klau [Kla01]. Besides these, there are more aesthetic requirements specified in the literature, especially Imhof [Imh75] and Yoeli [Yoe72] list some more. But the main requirements can be described as follows:

- **Unambiguity:** it shall be easy for the reader to see to which feature a label belongs,

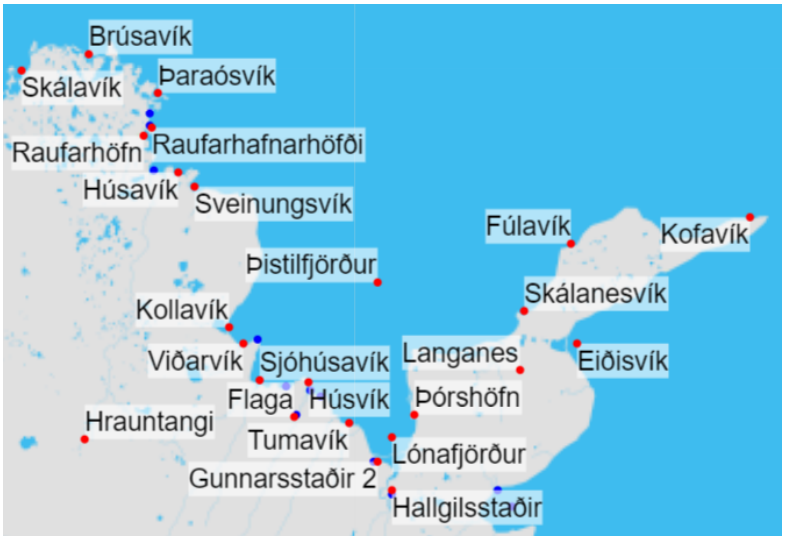


Figure 2.1: An example labeling of a part of Iceland. Features that have a label assigned (white box with text) are represented with a red filled circle. Features that are not labeled are shown with a blue filled circle.

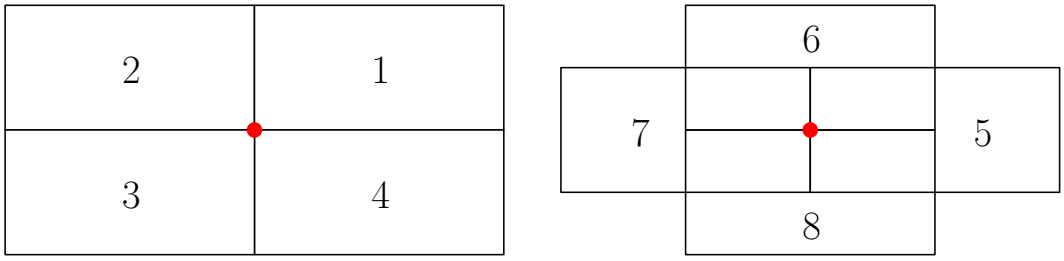
- **Legibility:** the information of the label shall be easy to read and fast to locate (the position or position arrangements with other labels also influences the legibility) and
- **Disturbance:** a label shall disturb or obscure other labels or features as little as possible (no or only a few overlaps).

Since this thesis omits line and area features but concentrates on point features only we give a formal definition of the point feature label placement problem (PFLP). We formulate it as follows:

Problem 1 (PFLP). *Given a map M and a set F of n point features (i.e., points of interest) in the Euclidean plane $\mathbb{R}^2$ that shall be labeled in map M. Each feature $f_i \in F$ $(i = 1...n)$ has a set C_i of possible label candidates (this set may be infinite). The goal of PFLP is to assign for each feature f_i at most one of the candidates of its set C_i so that certain objectives are optimally fulfilled. Examples of objectives are described in Section 2.2.*

Additionally we want to investigate approaches to preserve an existing labeling solution, e.g., after modification of some label candidates. Hence, a second problem arises that we define as follows:

Problem 2 (PFLP-Update). *Given a map M and a set F of point features in the Euclidean plane $\mathbb{R}^2$ that shall be labeled in map M. Let L be the set of all label candidates*



(a) (b)

Figure 2.2: Eight candidate positions for a point feature.

of all features $f \in F$. Further let $S \subset L$ be an existing labeling solution for map M. The goal of PFLP-Update is to calculate a new solution $S' \subset L$ so that $|S \cap S'|$ is maximum and certain objectives are optimally fulfilled.

Regarding the goal of our thesis we concentrate on the label placement on static maps where the zoom size is fixed and no map rotation is allowed. Panning is allowed in a "static" way, meaning that we only change the section of the map that is viewed without changing any label placement. Since we are mainly working on cartographic maps in this thesis we will concentrate on internal labeling with a discrete positioning model. Two examples for discrete positioning models that we use in this thesis are the *four-position* or the *eight-position* model.

- **Four-position model:** In the four-position model each feature has exactly four label candidates. Each candidate touches the feature in one of its corners. Figure 2.2a shows the four positions of the candidates.

- **Eight-position model:** The eight-position model extends the four-position model by the four candidate positions shown in Figure 2.2b. Each label candidate touches the feature exactly in the middle of one of its edges.

We define the following notations shown in Figure 2.3. A *point feature* represents a point of interest and is specified by its coordinates. Each point feature has a set of *label candidates* placed around the point feature. They are indicated by rectangles in Figure 2.3 but can be of any shape in general. If one of these candidates is selected for a solution we call it the *label* of the point feature (indicated by the solid line instead of a dashed line of the other label candidates). Each label candidate position is described by its *anchor point* which is always in the upper left corner of the bounding box of the label candidate. So the anchor point of the bottom right label candidate is equal to the point features coordinates.

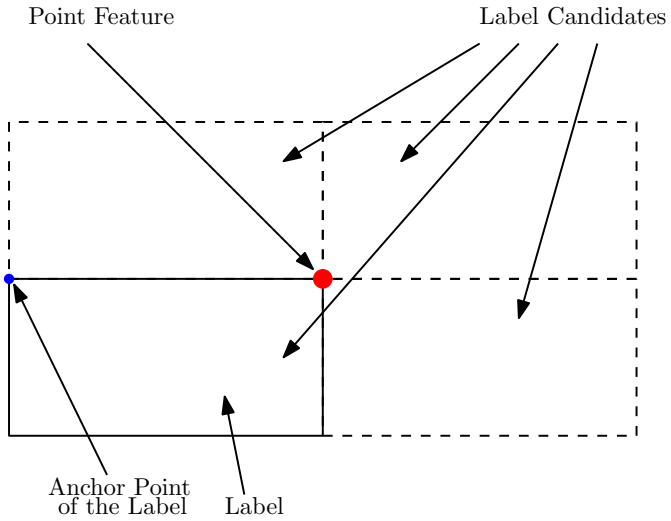


Figure 2.3: Notations used in this thesis.

2.2 Objectives in Label Placement

Dependent on what a cartographer or the creator of an illustration wants to achieve, different objectives can be formulated in map labeling. These objectives can mostly be described as an *objective function*. That is a function that assigns every possible labeling a value that describes the quality of it [CMS95]. Many subjective quality aspects and human expert knowledge in cartographic map creation or aesthetic requirements for good label placement like Imhof lists [Imh75] hardly can be formulated as an objective function. But quality aspects like the amount of overlaps, the number of features left unlabeled or even some a priori preferences (preferred label candidates) can be described formally and therefore also be used for automatic label placement approaches.

In the literature different objectives have been studied and formulated. Dependent on the pursued goal some of the proposed approaches result in a *conflict-free* labeling, i.e., a solution where no two labels overlap, others in a *conflict-minimal* labeling, i.e., a solution where all features are labeled and the number of conflicts between them is minimal. The main objectives used in the literature are defined as following problems:

Problem 3 (**Label Number Maximization Problem (LNMP)**). *The objective for this problem is to search for a subset in the set of all label candidates of all features that is maximum and conflict-free (i.e., there are no two labels in this subset that overlap each other or another feature). That means to try to place as many labels as possible with the permission to leave some features unlabeled [CRL08, WWKS01]. In terms of the objective function the optimal solution is the one with the highest value (i.e., largest amount of placed labels). A valid solution for the LNMP is conflict-free.*

Problem 4 (**Label Size Maximization Problem (LSMP)**). *The goal of the label size maximization problem is to find a maximum scale factor $\sigma > 0$ by which all labels*

(respectively label candidates) are stretched so that each feature is labeled and no two labels overlap in the corresponding solution [ArDT09, WWKS01]. Hence a valid solution for this problem is a conflict-free labeling, but the labels may become tiny.

Problem 5 (**Minimum Number of Conflicts Problem (MNCP)**)**.** *The MNCP problem was defined by Ribeiro and Lorena [RL08] but was also known as the Label Overlap Minimization Problem [Kla01, ArDT09]. Unlike the above mentioned problems in the MNCP label overlaps are allowed. The objective is to label all features without scaling them and at the same time minimize the total number of conflicts in the whole labeling. A valid solution for the MNCP is conflict-minimal.*

Problem 6 (**Maximal Number of Conflict Free Labels Problem (MNCFLP)**)**.** *The MNCFLP is another approach where all labels have to be labeled without stretching them. But the objective for this kind of problem is to maximize the number of labels that have no conflicts. Or in other words to minimize the number of labels that are obstructed by at least one other label [MRL10].*

Problem 3 as well as Problem 4 lead to solutions that are conflict-free and hence have no two labels in the solution that overlap. Problem 4 additionally ensures that all features are labeled like it is also in Problem 5 and Problem 6. The last two problems lead to solutions that are not free but minimal in their number of overlaps. All of the above mentioned problems can also be described in a weighted way, e.g., weights as priorities for features or label candidates, or weights for specific label size.

2.3 Terminology from Graph Theory

A *graph* is a tuple $G = (V, E)$ of a set V of *vertices* and a set E of *edges* $(u, v) \in V \times V$. The edges can have an orientation or no orientation and the graph is then called either a *directed graph* or an *undirected graph* respectively. In an undirected graph G an edge $(u, v) \in E$ with $u, v \in V$ is identical to the edge (v, u), i.e, the edges are unordered pairs of vertices. We say that two vertices $u, v \in V$ are *adjacent* to each other if they are connected by an edge $(u, v) \in E$. In an undirected graph this adjacency relation is symmetric.

A *clique* C in an undirected graph G is a subset of vertices $C \subseteq V$ such that all vertices in C are adjacent in G. A clique is called *maximum clique* if there is no other clique in G that is larger in cardinality than C.

A *vertex cover* D of an undirected graph is a subset of vertices $D \subseteq V$ such that for each edge $(u, v) \in E$ with $u, v \in V$ at least one of its endpoints is in D, i.e., $(u, v) \in E \Rightarrow u \in D \vee v \in D$. A vertex cover is called *minimum vertex cover* if it is of smallest possible size.

An *independent set* S of a graph G is a subset of vertices $S \subseteq V$ such that no two vertices $u, v \in S$ are adjacent in G. An independent set is *maximal* if it is not properly contained

in any other independent set of G. In a maximal independent set S each vertex u in the complement set $V \setminus S$ is connected by an edge $(u, v) \in E$ to some vertex $v \in S$, i.e., every vertex $u \in V$ either is in set S or has at least one adjacent vertex $v \in V$ in G that is contained in set S. A *maximum independent set* S is an independent set that is largest in its cardinality over all independent sets of graph G, i.e., there exists no independent set $S' \subseteq V$ containing more elements than S. Let $D \in V$ be a vertex cover of graph G. Then the complement $V \setminus D$ is an independent set of G.

Finding a maximum clique or independent set as well as finding a minimum vertex cover are classical optimization problems in computer science and are known to be NP-hard [GJ79].

CHAPTER 3

Data Structures

In this chapter we give an overview of data structures useful for automatic label placement. Section 3.1 describes the concept of quadtrees that are used for storing spatial data preserving their location. Section 3.2 describes the concept of conflict graphs that are used for describing conflicts between label candidates.

3.1 Quadtree

There is a high need on storing and indexing multi-dimensional data. This is not restricted to spatial data, as we need it for this thesis, but any multi-dimensional information that has to be stored or retrieved on composite keys, for example correlation data records where the two dimensions represent age and annual income. Hence, much research has been done to find efficient data structures. Even if the data are not spatial the multi-dimensional data records are often related to multiple dimensions in space where each dimension represents one attribute of the record [FB74]. Well known index structures for this kind of data are besides the *Quadtrees* [FB74] for example *Octrees* (the tree-dimensional analogon to the quadtree) [Mea82], *kd-trees* [Ben75] or *R-trees* [Gut84]. An often used approach is the R*-tree [BKSS90] which is a variant of the R-tree. Since the complexity increases with the number of attributes (=dimensions), techniques were developed to reduce high-dimensional datasets to low or medium-dimensional datasets (usually 8 or fewer dimensions) [KP07].

Especially when we look on spatial data, the datasets are usually very large and irregular distributed. When searching for specific data records or query over specific ranges of spatial objects, good indexing techniques are essential. Otherwise the whole dataset has to be searched to find spatial objects with a specific criterion which is unacceptable in practice for a large amount of data. For example if we have the set of all cities, towns and villages of Austria given by their longitude and latitude as the two keys (two-dimensional data records) a possible query could be to find all towns and villages that are within the

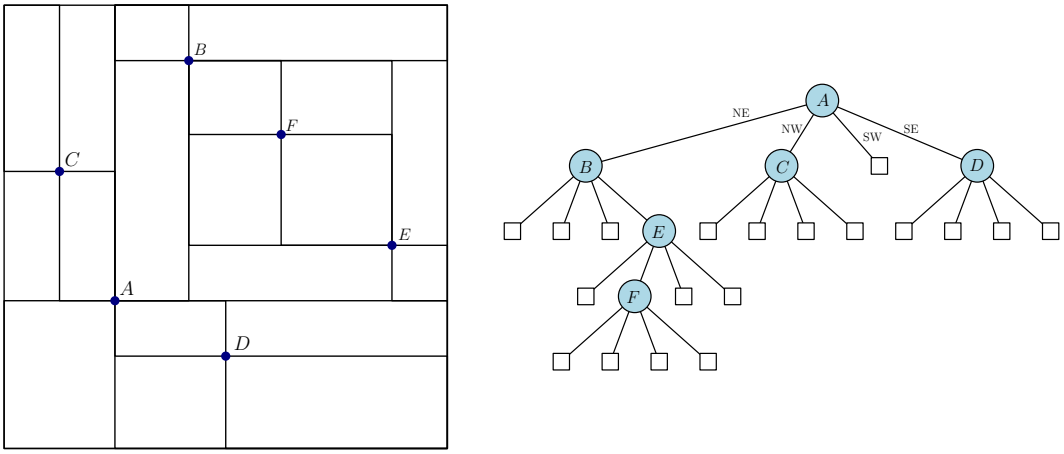


Figure 3.1: A point quadtree with subregions of different size and its tree representation.

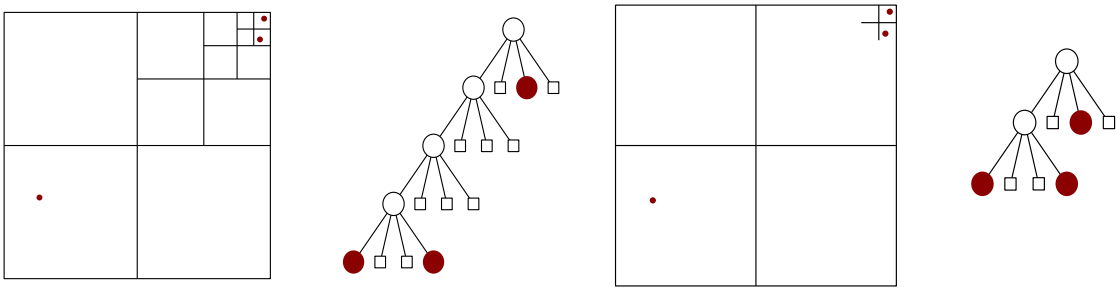
area of 50 kilometres to the north and 50 kilometres to the west of Vienna. This query shall be reasonable fast. Running through the whole set of places in Austria will be far to slow.

Another aspect a good indexing technique shall be able to handle is irregularly distributed data. In spatial datasets there can be many spots where the data are dense (e.g., cities and agglomerations) and on other regions the data are sparse or empty (e.g., oceans). Considering for example a *grid file* [NHS84] as an indexing structure there will be many overfull cells and other cells which are empty but also have to be stored. This space overhead shall be avoided in a good indexing technique.

A *quadtree* is a data structure first presented by Finkel and Bentley in 1974 [FB74]. The quadtree is a tree data structure for indexing two-dimensional data records and is commonly used for image processing [SB94], mesh generation [PM98] or spatial indexing with range queries (also known as *window queries*) [Sam90, AS95]. The term quadtree meanwhile has more than one meaning and actually describes a class of hierarchical data structures that are based on the principle of recursive decomposition of space. They differ in the type of data they represent (e.g., points, rectangles or curves), the resolution (describes the number of decompositions and can be variable or fixed) or the process of decomposition, i.e., decomposition into subregions of equal size (quadrants) or regions of different size. If the subregions for one level are all of the same size it is called regular decomposition [Sam90].

As the name suggests, in a quadtree the space is partitioned by recursively subdividing it into four subregions (also called *quads*, *cells* or *buckets*). In general these regions can be squares or rectangles and do not need to have the same size within a quadtree [FB74]. To ease the denotation of the subregions they are often described in the map analogy of northeast (NE), northwest (NW), southwest (SW) and southeast (SE).

Now looking on the tree representation, a quadtree is a rooted tree where each node is



(a) low node occupancy

(b) compressed quadtree

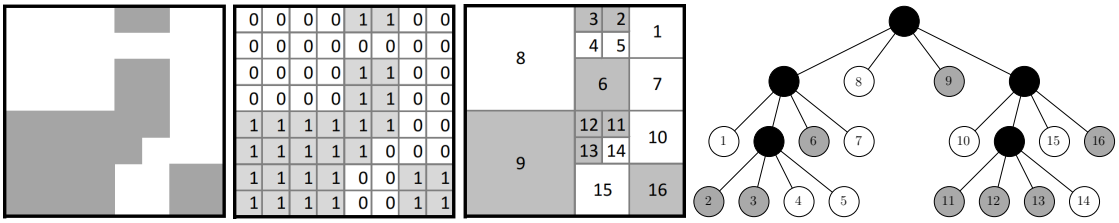
Figure 3.2: An unbalanced point region quadtree (a) and a compressed quadtree (b) based on the same input set.

either an *internal* node with exactly four children or a *leaf* node with no children (the root node covers the entire area). Figure 3.1 shows a general *point quadtree* (further described in Section 3.1.1) with quads of different size and its corresponding tree representation. We define the ordering of the children in the tree representation as follows: the NE quad is the first child, the NW quad is the second, the SW quad the third and the SE quad corresponds to the fourth child.

Unlike other index structures (e.g., B-tree or R-tree) quadtrees are unbalanced. This is on the one hand an advantage when it is used with data where a lot of balancing would be necessary using a balanced data structure. On the other hand the unbalanced property of quadtrees can be a drawback because the tree can be fanned out and there can be a lot of not occupied (i.e., empty) nodes that also have to be stored. Figure 3.2a shows a quadtree with low node occupancy. It shows an extreme case of a *point region quadtree* (described in more detail in Section 3.1.1) with a maximum capacity of one point per cell. It contains three points which are located in a way that the quadtree has to be partitioned a lot and that one deep branch is created while most of the leaves are empty. But this artefact can be eliminated by efficient node packing techniques like the *compressed quadtree* [HP05] or the *skip quadtree* [EGS05] which is actually a *region quadtree* (described in Section 3.1.1) based on a compressed quadtree. Figure 3.2b shows a compressed quadtree based on the same dataset of the point region quadtree shown in Figure 3.2a.

An advantage of the quadtree is the disjoint and regular space partitioning strategy. It ensures that there are no overlaps between the quadrants. Different to this is the R*-tree which does not result in disjoint decomposition of space and suffers from overlaps of the MBRs (minimal bounding rectangles) even for low dimensional data [KP07].

An important approach in literature is the *pyramid technique* introduced by Berchtold et al. [BBK98]. Like the quadtree it is an indexing technique that partitions space into disjoint regions and can be used even for data of higher dimensionality. In the pyramid technique data space is first divided into $2d$-pyramids which have their top in the center of the space and secondly these pyramids are cut into slices parallel to their basis. With



(a) (b) (c) (d)

Figure 3.3: (a) An image, (b) its binary array representation, (c) its decomposition to maximal blocks (d) and its tree representation.

this partitioning strategy d-dimensional data points are transformed to 1-dimensional values (keys) which then can be stored in any efficient one-dimensional order-preserving index structure (e.g., B^+-trees). The keys are stored together with the corresponding d-dimensional points in the leaf nodes [BBK98]. Kim and Patel [KP07] showed that the pyramid technique outperforms other indexing structures on higher dimensions but performs worse than quadtrees for low dimensions. Additionally they showed that the quadtree performs much better than R*-trees and the pyramid technique on skewed data (i.e., data with areas where data points are dense and other where there are no points).

Gargantini [Gar82] introduced an effective representation of quadtrees called *linear quadtree* which is said to save 66 percent of storage required by a regular region quadtree. The linear quadtree encodes only the non-empty quads as integers and stores them in a sorted array. Samet et al. [SRSW84] further improve this quadtree and use the B-tree indexing structure instead of an array and therefore benefits from the efficient and well performing B-trees.

3.1.1 Common Types of Quadtrees

As described above there is not "the one" quadtree but the term describes a class of data structures. In this section some common types of quadtrees shall be described. This list is not to be regarded as exclusive. A more comprised list of quadtrees and other hierarchical data structures can be found in [Sam84].

Region Quadtree

The region quadtree is the most common quadtree representation. It partitions the two-dimensional space into four equal-size squares as long as more refinement is desired. So the depth of the quadtree is dependent on the given input data. The sides of the squares are always a power of two long and the value stored in a node is always applied to the whole area of the quad [Sam90]. Since the region quadtree often represents an image array, let us explain the region quadtree with the image shown in Figure 3.3a. It shows an image with size $2^3 \times 2^3$ pixels. Figure 3.3b shows the binary array representation of this image. A pixel value of 1 means that it belongs to the picture element (the gray

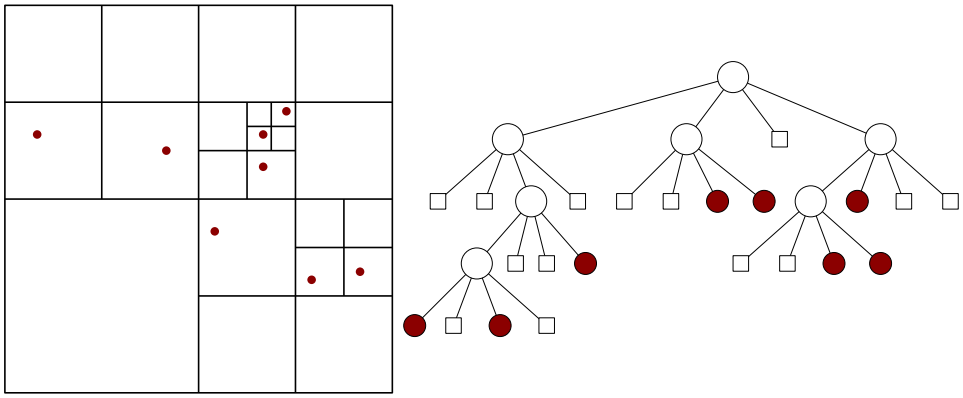


Figure 3.4: A point region quadtree with equal size quadrants and its tree representation.

image region) and a pixel value of 0 means that it does not belong to it. When creating the region quadtree it is partitioned until there are only 1s or 0s in one quad. This can go down until the block size of 1×1 (one pixel). The optimal partitioning with maximal blocks is shown in Figure 3.3c and the resulting region quadtree of the image can be seen in Figure 3.3d. The gray and white nodes represent the corresponding regions in the image while the black nodes represent inner nodes.

Point Region Quadtree (PR Quadtree)

The PR quadtree is an adaption of the region quad tree and is intended to store point data. The difference is that in a region quadtree the value stored in a leaf node applies to the entire area of the node while the leaf nodes of the PR quadtree store a list of points that lie within the boundings of the corresponding quad. Each node has a maximum capacity for storing points. When this capacity is exceeded (e.g., when inserting a new point into a full quad) the quad gets divided into four new subquads and all the stored points of this quad are put to a proper subquad. Figure 3.4 shows the equal-size decomposition of a point region quad tree with maximal capacity of one point per quad. The point data are only stored in the leaf nodes indicated by the red nodes. Empty nodes are represented by white squares. The depth of the tree depends on the spatial distribution of the points and therefore the quadtree can be quite unbalanced. Remember the point region quadtree shown in Figure 3.2a. The tree could even be worse if the two points on quadtree level four are much closer together. So the depth of this kind of quadtree can not be expressed in the number of points it stores but as de Berg et al. [dBvKOS00] describe, the depth is related to the distance between the points and the size of the initial quad (the boundings of the root node). The depth is at most $log(s/c) + \frac{3}{2}$, where c is the smallest distance between any two points and s is the side length of the root quad. De Berg et al. [dBvKOS00] showed a proof for that.

Point Quadtree

The point quadtree is an adaption of the binary search tree to two dimensions and is used to represent multi-dimensional point data. This type of quadtree divides its quads in rectangular subregions that do not need to be square as it is for the above mentioned quadtree types. Every time a point is added to the point quadtree a search is done first to check if the point already exists in the tree. If not then the point is added to the appropriate quad (a leaf node) which is then divided in four new subregions with the point as the center of the subdivision. So the inserted point has four empty child nodes. The depth of the tree is strongly dependent on the insertion order. So if we insert a set of points in a "bad" ordering the depth can be linear in the number of input points.

Figure 3.1 shows the different size of the subregions in a point quadtree. The figure on the right side shows the tree representation. Each inner node contains exactly one of the input points. The leaf nodes are empty and also have to be stored. The point quadtree was improved and surpassed by the kd-tree [Ben75].

3.1.2 Operations on the Quadtree

Since the operations are different on various types of quadtrees we will consider here the operations on a point region quadtree because we use it in our thesis. The point region quadtree may be not the fastest or most efficient quadtree for storing the spatial data we need in our thesis, but it is a simple approach that satisfies the requirements of our thesis. Other approaches like the linear quadtree, skip quadtree or even the compressed quadtree will not use that much memory space and may perform better (which we actually did not prove since it lies outside the realm of this thesis). But it is no main goal of this work to find the best data structure for storing the spatial data. However, the framework is implemented in a way so that the data structure can be exchanged by a better performing structure very easily.

Insertion

Since the quadtree is unbalanced insertion into a PR quadtree is straight forward and is similar to the insertion in ordinary binary search trees [FB74]. Starting at the root, each of its child nodes is checked if the point we want to insert lies within its boundaries. If the point lies outside of the boundings of the root quad it can not be added to the quadtree. Since the quads are disjoint only one child node is selected at each level. This is done recursively until we find the node respectively the quadrant the point belongs to. Since a point can only be inserted into a leaf node the search goes down until a leaf node is reached. The point is then added to the point list of this node, given that the capacity is not exceeded. If there is no space in this list the quad is recursively subdivided into four subquads until the point can be added. The points of the parent node are distributed to the proper child nodes.

For the point region quadtree it is irrelevant in which order the points are added. The resulting tree will always be the same. In case of "bad" datasets (i.e., many points are

located close to each other and others are far away from them) the quadtree may get subdivided very often and may result in a completely unbalanced tree. As mentioned before the depth of the PR quadtree can not be described in the number of points it stores but in the distance between the points and the size of the initial quad. So the complexity for inserting a point into the quadtree is in $O(log(s/c))$ where c is the smallest distance between any two points including the point we want to insert and s the side length of the root quad. But in the case of a well distributed dataset and therefore a well balanced quadtree a point can be inserted in $O(log\ n)$ time where n is the number of points stored in the quadtree.

Deletion

The deletion of a point is the counterpart of the insertion and works similar to it. Instead of subdividing a quad it may require that some nodes are merged. First we try to find the point. Therefore we start at the root and go down the tree like we do for insertion until we reach a leaf. Again there can only be one leaf that contains the point, since the quads are disjoint. Arrived at the leaf we look into its list of stored points. If it does not contain the point nothing is done and the point is considered as deleted. Otherwise we delete the point and check if we have to merge.

Let v denote in the following the node where the point was deleted from. Then merging has to be done if the sum of all points stored in the subtree rooted in the parent of v is less than or equal to the capacity of one node. If we need to merge we first go up to the immediate parent of v and check if all the other children are leaves. If not then we cannot merge because a child contains a subtree and v is kept as empty leaf node. If all children of the parent are leaves we check if the sum of points stored in the leaves exceed the capacity of the tree. Only if the sum is less or equal to the capacity we can merge and move all stored points to the parent node. The child nodes are not used any more and are deleted. If merging was successful we need to recursively check the parent nodes to see whether we can continue merging until a merge is no longer possible.

The complexity for the deletion of a point from the point region quadtree is $O(d)$, where d is the depth of the quadtree. Its just a point search with subsequent deletion of it.

Point Search

Searching for a specific point is as simple as going down the correct branch deciding on every inner node in which child node the point lies that we want to find. This is simply done by comparing the coordinates of the point with the bounding box of the quad each quadtree node represents. When we reach a leaf node we just have to look into the list of points stored in this node if we can find it. If not then the point does not exist in the quadtree. Searching for a point is in $O(d)$ where d is the depth of the quadtree.

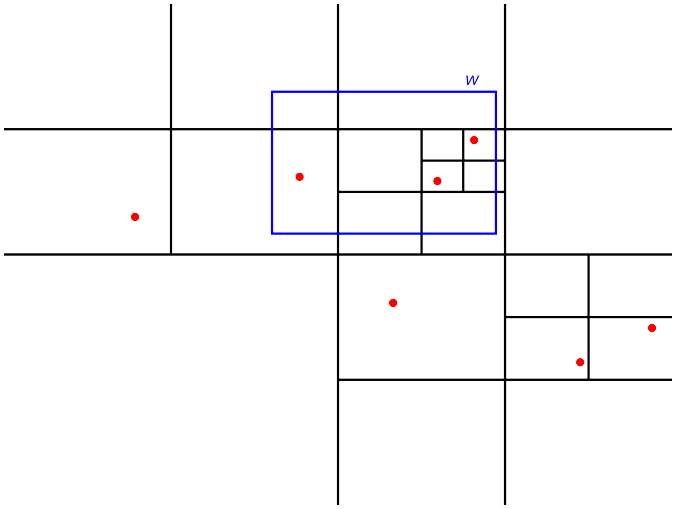


Figure 3.5: A window (blue rectangle) in a point region tree.

Region Search (Window Query or Range Query)

A typical query on multi-dimensional data is to request all records that meet two-dimensional criteria, i.e., records that lie within a specific range (also called *window*). Thus a window can formally be defined as $w = \{(x, y) \mid (x, y) \in [x_1 : x_2] \times [y_1 : y_2]\}$ where the interval from x_1 to x_2 describes the first dimension and interval y_1 to y_2 the second dimension [AS95]. Taking up the example of correlation data records combining age and annual income mentioned at the beginning of this section, a range query can be to find all persons that are between 30 and 40 years old and have an annual income between 10,000 and 20,000 euro. The resulting window can be described with $w = \{(x, y) \mid (x, y) \in [30 : 40] \times [10,000 : 20,000]\}$. Figure 3.5 shows a possible window w in a point region quadtree.

This window (in the following denoted as "range") is then used by the range query algorithm which can be seen in Algorithm 3.1. Starting at the root the initial call for this recursive procedure will be $QueryRange(range, root)$. We first initialize an empty list L where we want to gather all points that lie within the range. We then check if the boundary of the current quad intersects (or includes) the given range r (line 2). If not then return and ignore the whole subtree since no node in the subtree will intersect r. If we have an intersection detected (line 5) we test if we are in a leaf node and if true we add all points to our list L that are located within r. Else if we are currently in an inner node we recursively perform the query range on all four child nodes (NE, NW, SW and SE) and add their returned list to our list L (line 13). When the recursion returns we have received a list of all points within r.

All in all the region search works like the point search but with the difference that we possibly have to go down in several subtrees to get all points that lie within the window. The advantage of the quadtree data structure respectively of a region search over other

indexing structures is, that in general many records stored in the tree do not need to be examined.

The worst case is if the window is as big as the bounding box of the root node. Hence we have to examine all subtrees (i.e., all nodes). As the number of nodes is in $O((d+1) \cdot n)$ [dBvKOS00], where d is the depth of the quadtree and n the number of points stored in the quadtree, the range query can be done in worst case in $O((d+1) \cdot n)$ time.

Algorithm 3.1: QueryRange(range r, node n)

Input: A two-dimensional range r and a node n on which the query is applied.
Output: A list L containing all points within r.

```
1  L = ∅;
2  if boundary of this quad does not contain or intersect r then
3  |  return L;
4  end
5  if this node is a leaf then
6  |  for p ← points stored in n do
7  |  |  if p lies within r then
8  |  |  |  add p to L;
9  |  |  end
10 |  end
11 else
12 |  for child ← children of n do
13 |  |  L ← L ∪ QueryRange(r, child)
14 |  end
15 end
16 return L;
```

3.2 Conflict Graph

Many approaches for the label placement problem (exact or heuristic) have their solution calculations based on *conflict graphs*, especially when using a discrete positioning model where the positions and the number of label candidates are fixed. The first formulation of a conflict graph was done by Kakoulis and Tollis [KT98]. They described a graph, called *overlap graph*, that represents the relation between features and label candidates as well as label candidates among themselves. It contains a node for every feature and every label candidate and an edge for those who are related to each other. Most other researchers in literature omit the relations between features and their candidates but only keep the relations between label candidates in their conflict graph [MRL10, WWKS01, CRL08].

Formulating the labeling problem with a conflict graph revealed the close relation to the independent set or more specific to the maximum independent set problem. Based on

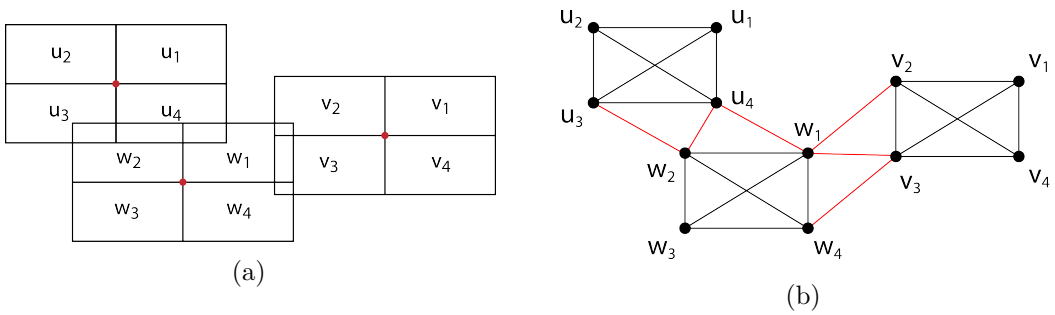


Figure 3.6: Three point features and their corresponding conflict graph.

this approach much research was done (e.g., [SVA00, WWKS01]).

Strijk et al. [SVA00] show a reduction from the map labeling problem to the maximum independent set problem and describe different heuristics that are based on finding a large or optimal independent set as well as heuristics for the map labeling problem itself. Additionally they describe a branch-and-cut optimization algorithm and compare and evaluate it with the heuristics on randomly generated map labeling problem instances.

But using a conflict graph is not restricted to solving the maximum independent set problem. Cravo et al. [CRL08] applied a *greedy randomized adaptive search procedure* (GRASP) to the point feature label placement problem based on the conflict graph structure. GRASP is a metaheuristic first introduced by Feo and Resende [FR89] and is used to find approximate solutions to combinatorial optimization problems. It is an iterative process where each iteration consists the two phases of construction and local search. In the first phase a greedy randomized solution is constructed which is then improved in the local search phase to find a local optimum. The best solution over all iterations is kept [RR10].

Formal Definition of the Conflict Graph

Given a set F of graphical features to be labeled, and a set L of label positions for all $f_i \in F$, the conflict graph $G = (V, E)$ is defined as follows:

- $V = \{v_1, v_2, ..., v_n\}$ where $n = |L|$ and every v_i represents a candidate $l \in L$
- $E = \{(v_i, v_j) \mid v_i, v_j \in V,\ i \neq j,\ v_i \text{ and } v_j \text{ overlap}\}$

In other words the conflict graph contains for each label candidate $l \in L$ a node in the set of its vertices V. Therefore the number of nodes in the conflict graph is equal to the number of label candidates ($|V| = |L|$). The set of edges E contains for all node pairs $v, w \in V$ that overlap each other (i.e., their bounding boxes intersect or one box lies within the other) an edge (v, w) which is equal to (w, v) since the conflict graph is undirected. Note that a node can not be in conflict with itself.

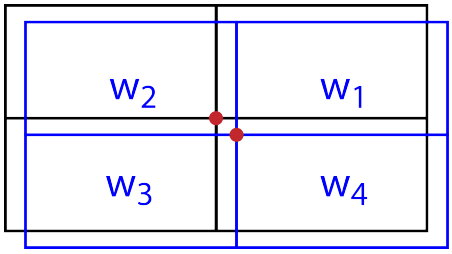


Figure 3.7: The maximal possible overlap of two point features with the fixed four position model.

Figure 3.6a illustrates three point features (u, v and w) and their label candidates (u_i, v_i and w_i with $i = 1..4$) in a fixed four position model. The corresponding conflict graph is shown in Figure 3.6b. Each label candidate is in conflict with all other candidates of the same feature (i.e., the candidates of each feature form a clique in the conflict graph). This is indicated by the black edges. The overlappings of w_2 with u_3 and u_4, of w_1 with u_4, v_2 and v_3 as well as the overlapping of w_4 with v_3 are represented in the figure as red edges.

Complexity

Assuming a discrete position model with a fixed number of possible positions, the number of nodes in a conflict graph $G = (V, E)$ is always in $\Theta(n)$ where n is the number of features (regardless of the number of overlaps) or more precisely the number is always $c \cdot n$ where c is the number of possible positions per feature. The complexity for the edges is different: assuming that no label candidate of one feature overlaps a label candidate of another feature, i.e., there are no two nodes $v_{i,k}$, $v_{j,l} \in V$ so that $(v_{i,k}, v_{j,l}) \in E$ where i and j are indices of two features f_i, $f_j \in F$ (with $i \neq j$) and k and l are indices of the label candidate of feature f_i respectively f_j. Then the number of edges of G is in $\Theta(n)$ too.

But considering the worst case where every feature overlaps every other which is actually unrealistic but possible, e.g., consider labeling all bars and restaurants of a city on a map with a very low zoom level, the number of edges grows fast. Figure 3.7 shows two point features in a fixed four position model that have a maximal overlap (regardless if they have label candidates of same size or not). So additionally to the six overlaps the label candidates of both features have separately, w_4 has one, w_1 and w_3 have two and w_2 has four more overlaps that sum up to nine new overlaps. We can formally describe this with $6n + 9\sum_{i=1}^{n-1} i$ where n is the number of point features and $\sum_{i=1}^{n-1} i$ describes the maximum number of conflicting point features. We multiply this sum by nine because as we have seen, two point features can have maximum nine conflicts of their label candidates. Simplifying then leads to $6n + 9 \cdot \frac{n^2-n}{2} = \frac{9n^2+3n}{2}$. Thus the number of edges in this worst case for the fixed four position model can be described by the explicit function $g(n) = \frac{9n^2+3n}{2}$ where n is the number of features. So the number of edges grows

in worst case quadratic and is in $O(n^2)$.

Hence the conflict graph can get very large and complex even for low numbers of features so that it is hard to deal with it. For example when we have a set of thousand features that overlap each other there can already be a million edges. Wagner et al. [WWKS01] presented an approach of three rules that reduce the size of a given conflict graph without influencing the optimality of the solutions. The basic idea is to apply the three rules exhaustively to all given features to label as many as possible and at the same time reduce the number of label candidates of other features and thus removing edges from the conflict graph.

CHAPTER 4

Labeling Algorithms

This chapter covers different approaches for automatic map labeling. We use algorithms for the point feature label placement problem proposed in the literature or adapt them to our labeling problem. All approaches presented in this chapter assume a discrete positioning model and most of them are based on a conflict graph described in the previous chapter. The algorithms are intended to solve the point feature labeling problem, but they are not necessarily restricted to point features only. More precisely, if we use a conflict graph as the basis for our algorithm we are always independent of the type of the features we want to label.

4.1 Maximum Independent Set Approaches

The maximum independent set problem is a well-known problem in graph theory where the goal is to find a maximum subset of vertices in a graph so that no two vertices contained in the set are adjacent. As the map labeling problem can be described with a conflict graph where the vertices represent label candidates and the edges represent the conflicts (i.e., intersections) between them, the similarity can be seen easily. This is since in map labeling (regarding the label number maximization) it is necessary that no two adjacent vertices shall be labelled at the same time and thus a valid solution shall form a pairwise disjoint set, or in other words, form an independent set. For the label number maximization problem the task is to find a maximum independent set. However, it is known that computing a maximum independent set is NP-hard and thus approximation algorithms or heuristics are necessary for larger instances.

Agarwal et al. [AvKS98] present in their paper an approximation algorithm for labeling point features with rectangular labels of fixed but different size as well as an approximation scheme for labels with unit height and varying width. Their polynomial-time approximation algorithm is based on divide and conquer and computes a large independent set in a set of rectangles in the plane.

Strijk et al. [SVA00] show a reduction from the map labeling problem to the maximum independent set problem and describe different heuristics that are based on finding a large or maximal independent set as well as heuristics for the map labeling problem itself. Additionally they describe a branch-and-cut optimization algorithm and compare and evaluate it with the heuristics on randomly generated map labeling problem instances.

Verweil and Aardal [VA99] present a branch-and-cut algorithm for finding maximum independent sets in the conflict graph and apply it to instances of map labeling. They incorporated a local search heuristic in their algorithm to improve the quality of the solution found by the branch-and-cut part.

More recent approaches use reductions to reduce the input size for the maximum independent set problem while maintaining the optimality of the solutions. These reductions turned out to be critical for efficient maximum independent set algorithms [Str16]. Some approaches are based on *kernelisation* [GD06, AKCF$^+$04], which is a polynomial-time preprocessing stage in which the input to the algorithm is reduced to a smaller input, called the *kernel*. The output of an algorithm on the reduced input shall be the same or shall be easy to transform to the output for the original problem. Kernelization is often used in algorithms that solve the *(minimum) vertex cover problem*, which is the complementary problem to the (maximum) independent set problem. So if C is a minimal vertex cover of a graph $G = (V, E)$, then $V \backslash C$ is a maximum independent set of the graph. Hence, when we solve the minimum vertex cover problem for a graph we can simply transform it to a maximum independent set.

Butenko and Trukhanov [BT07] showed a reduction based on critical independent sets which are solvable in polynomial time, and demonstrated their approach to solve the maximum independent set problem on large graphs with up to 18,000 vertices. Very promising is the work of Akiba and Iwata [AI16]. They showed that applying advanced reductions is highly effective in practice and that an exact minimum vertex cover, and hence an exact maximum independent set, can be found in graphs with up to 2 million vertices. But they focus on sparse graphs of real networks (e.g., social networks, web graphs or road networks).

Strash [Str16] showed that only two simple reductions are sufficient for many real-world instances namely *vertex folding* and *isolated vertex*, and that the power of advanced rules comes largely from their initial application (i.e., kernelization) and not their repeated application during branch-and-bound.

The reductions are always done as a preprocessing step. Since these techniques seem to be very promising it would be nice to see what impact such reductions would have on our labeling instances in our framework. But this exceeds the realm of this thesis and implementing reductions is future work.

Priorities

If every vertex has a (possibly different) weight, meaning that each vertex has a different importance or priority, then we can build a *maximum weight independent set*. This is an

independent set where the sum of the total weights of the vertices in the independent set is maximum. The maximum independent set problem is a special case of the maximum weight independent set problem where all weights are one.

4.2 SAT Approaches

In propositional logic a *literal* is a Boolean variable p, called positive literal, or its negation $\neg p$, called negative literal. A *logical operator* is a symbol to logically connect two or more literals or formulas, e.g., the conjunction ("AND", denoted by the symbol $\wedge$) or the disjunction ("OR", denoted by the symbol $\vee$). A formula ϕ is a set of literals that are connected by logical operators. A satisfying *truth assignment* for a formula is an assignment of truth values to its propositional variables such that the whole formula evaluates to true. A *clause* is a disjunction of literals, i.e., all literals are connected with the logical operator "OR". A formula is in *conjunctive normal form* (CNF) if it is a conjunction of one or more clauses. Every formula ϕ can be transformed to an equivalent formula that is in CNF. If every clause consists of exactly two literals, then the formula is of the type *2-SAT* and can be evaluated in time proportional to its length. Finding an assignment for any other k-SAT problem with $k > 2$ is NP-complete.

The Boolean satisfiability problem (also called SAT) is well known in computer science and is the problem of determining if there exists an interpretation (i.e., a truth assignment) that satisfies a given Boolean formula. It is a NP-complete decision problem and is well researched. Hence there are many practically good and fast SAT solvers (i.e., algorithms that decide if a given Boolean formula has a truth assignment) developed for these decision problem.

In the area of map labeling for example Formann and Wagner [FW91] or Marks and Shieber [MS91] studied the labeling problem in relation to the SAT problem. Both have proved it to be NP-complete. Forman and Wagner researched the following point labeling problem which they call the packing problem. Given a set of points in the plane, label them with axis-parallel equal-sized square labels and make them as large as possible so that no two labels overlap and that each point is labeled. They used the four-position model where the point lies in one of the corners of the labels. Each label candidate is seen as a Boolean variable and for each pair of overlapping candidates a clause is generated. The set of clauses is then checked if there is a satisfying truth assignment.

Formulation

Since we have the conflict graph as a basis for our labeling approach it is quite straight forward to translate the conflicts to SAT and generate a corresponding 2-SAT formula in conjunctive normal form. Let us formulate our labeling problem as follows.

For a given conflict graph $G = (V, E)$ it holds that there is an edge $(u, v) \in E$ between two nodes $u, v \in V$ if and only if they are in conflict with each other (i.e., the label candidates overlap). Thus, introducing two Boolean variables p_u and p_v and associate

them to the label candidates u and v we can build the clause $\neg(p_u \wedge\ p_v) = \neg\ p_u \vee \neg\ p_v$. This indicates that we do not want that p_u and p_v are true at the same time meaning u and v are not both set as label for their feature simultaneously since there is a conflict between them in the conflict graph. A Boolean value of 1 (=true) means in our case that the label candidate is assigned to the feature as its label. The Boolean value of 0 (=false) means the opposite.

Now generating for every edge in E such a clause we can generate the Boolean formula as follows.

$$\bigwedge_{(u,v)\in E} (\neg p_u \vee \neg p_v) \tag{4.1}$$

It is easy to see that the formula is true if $p_u = 0$ for all $u \in V$ meaning that all Boolean variables are set to false and thus no point feature gets a label. But this is not what we want, since we want to maximize the number of labeled features.

The Way to Maximization

Hence we have to go beyond this and improve our approach by trying to maximize the number of positive literals. For this we reformulate our labeling problem as a *maximum satisfiability problem* (MAX-SAT) which is a generalization of the classical Boolean satisfiability problem. MAX-SAT is an optimization problem of determining the maximum number of clauses that can be satisfied for a given Boolean formula in conjunctive normal form.

Now adding for each node $u \in V$ a clause with the single literal p_u indicating that we want the literal to be positive, we get the formula shown in Equation 4.2. In conjunction with Equation 4.1 it then is solved as MAX-2-SAT problem. While the 2-SAT problem can be solved in polynomial time in the product of the number of clauses and the number of variables in the given instance, the MAX-2-SAT problem is NP-hard [GJ79].

$$\bigwedge_{u\in V} p_u \tag{4.2}$$

The issue that arises here is that a clause representing a conflict between two nodes may now be violated, see Table 4.1. It shows a very simple example of the Boolean variables of two label candidates r and s that overlap (i.e., $\neg r \vee \neg s$ is in the set of clauses) and all possible truth assignments. The last line counts the number of positive clauses. As can be seen the assignment $r = s = 1$ is also maximal but the very important clause $\neg r \vee \neg s$ is violated. Thus we need some priorities for the clauses that represent the conflicts of the conflict graph. Here the *partial maximum satisfiability problem* (PMAX-SAT) can do the trick. The PMAX-SAT problem subdivides the set of clauses in a set that has to be satisfied (*hard* clauses) and a set where the positive clauses are maximized (*soft* clauses) or in other words the number of falsified soft clauses is minimized [ABL10]. In our case the conflict clauses are hard clauses and the clauses in Formula 4.2 are soft

r	0	0	1	1
s	0	1	0	1
$\neg r \vee \neg s$	1	1	1	0
Sum	1	2	2	2

Table 4.1: Three clauses, their truth assignment and the sum of positive clauses for a single conflict.

clauses. Partial MAX-SAT is between MAX-SAT where no clause has to be inevitably satisfied (i.e., all clauses are soft) and SAT where all clauses have to be satisfied (i.e., all clauses are hard). The PMAX-SAT problem is NP-hard [ABL10].

Priorities and Weighted Partial MAX-SAT

In PMAX-SAT the soft clauses can be assigned different weights which can be seen as the penalty for falsifying the clause. This makes the PMAX-SAT instance weighted. The aim of weighted PMAX-SAT is to find an assignment that satisfies all the hard clauses and minimizes the sum of penalties of the falsified soft clauses. The idea behind giving different weights to soft clauses is that not all clauses are equally important. Hard clauses can be seen to have infinite weight (i.e., maximal importance) [ABL10]. A practical simple value for the weight of hard clauses is the sum of the weights of all soft clauses plus one. So falsifying one hard clause has a penalty higher than it would be when falsifying all soft clauses.

In our thesis we use the weighting of soft clauses on the one hand for priorities of some label candidates. Higher priority means a higher weight respectively a higher penalty and hence it is more likely that the clause is satisfied. This automatically means that the objective of a label number maximization is changed to a maximization of the weights of all labels in the solution. On the other hand we use weights of soft clauses to preserve an initial solution (see Problem 2), e.g., for recalculations after a label modification, where the soft clauses representing the labels in the initial solution have a slightly higher weight than the others. The question that arises is: How can we preserve the current solution as much as possible but also try to maximize the elements in the solution, i.e., maximize the number of labeled features? The answer lies in a correct weighting of the soft clauses.

Assume that all label candidates have the same priority. The weighting of the clauses for the recalculation can be done as follows: Hard clauses remain almost unchanged in their weight which is still the sum of weights of all soft clauses. To achieve our goal that the labels of the initial solution are also contained in the newly calculated solution, we have to adapt the weights of the soft clauses depending whether they were contained in the initial solution or not. Let w denote the weight of a soft clause in the initial solution with k elements. Then the new clauses for label candidates that were not contained in the solution still have a weight of w. Clauses contained in the initial solution which we want to preserve, get a weight of $w + \varepsilon$ with $\varepsilon > 0$ assigned so that the chance is higher that

they get satisfied in the new solution because of the higher penalty. We can not choose ε arbitrarily since we still prefer a maximum solution to a solution that is similar to the initial solution. Consequently we have to choose an $\varepsilon > 0$ so that $(k-1)(w+\varepsilon) < k \cdot w$ or simplified $\varepsilon < \frac{w}{k-1}$. This indicates that we prefer a solution with maximal elements to a solution that is similar to the initial solution but with one or more elements less.

Now allowing the candidates to have different priorities we can formulate the above constraint as sums over the weights with $\sum_{i=1}^{k-1}(w_i+\varepsilon) < \sum_{i=1}^{k} w_i$ which can be simplified to $\varepsilon < \frac{w_r}{k-1}$ with $r \in \{1, 2, .., k\}$ where w_r is the weight of a possible removed label candidate from the initial solution.

If we choose $\varepsilon < \frac{w_{min}}{k-1}$ with $w_{min} = \min(w_1, ..., w_k)$ then recalculating a solution after a label modification and preserving the previous solution as much as possible works with weighted PMAX-SAT for instances with and without priorities. The only drawback is that after a priority modification of a label this approach will fail and we can neither guarantee that the number of labels is maximized nor that the previous solution is preserved as much as possible. Then automatically the sum of weights of the labels in the solution is maximized.

4.3 Three Rules Algorithm

The *three rules algorithm* is an approach proposed by Wagner et al. [WWKS01] and aims for label number maximization. The idea of this algorithm is to separate the geometric and the combinatorial parts of the labeling problem. It therefore uses the conflict graph of the label candidates on which a set of rules is applied to simplify it without influencing the size of an optimal solution. The advantage of this algorithm is that it does not depend on the shape of the labels and can be applied to label point, line and area features given that each feature has a precomputed finite set of label candidates. It is quite easy to implement, runs fast and returns good results in practice as they show in their paper [WWKS01].

The algorithm consists of two phases:

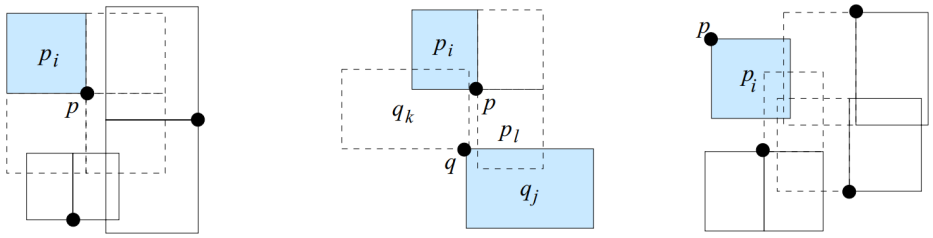
- Phase I applies the set of rules exhaustively to all features.
- Phase II is a heuristic that eliminates candidates as long as each feature has at most one candidate left.

This algorithm uses three rules that are designed for rectangular, axis-parallel label candidates. The number of label candidates per feature is not restricted but for simplicity they are described for the fixed four position model. In the following rule definition p_i describes the ith label candidate of feature p.

The illustrations (Figures 4.1 a to c) shall help to understand the three rules and show the label candidates that are chosen to label a feature in blue shading and the candidates

that are deleted after the rule was applied are drawn with dashed lines. The following rules are taken from Wagner et al. [WWKS01]:

- Rule 1: If p has a candidate p_i without any conflicts, declare p_i to be part of the solution, and eliminate all other candidates of p, see Figure 4.1a.
- Rule 2: If p has a candidate p_i that is only in conflict with some q_k, and q has a candidate q_j $(j \neq k)$ that is only overlapped by p_l $(l \neq i)$, then add p_i and q_j to the solution and eliminate all other candidates of p and q, see Figure 4.1b.
- Rule 3: If p has only one candidate p_i left, and the labels overlapping p_i form a clique, then declare p_i to be part of the solution and eliminate all labels that overlap p_i, see Figure 4.1c.



(a) Rule 1. (b) Rule 2. (c) Rule 3.

Figure 4.1: Three rules to simplify a conflict graph. Images taken from [WWKS01]

Phase I.

In phase one the three rules are applied to all features exhaustively to label as many features as possible and to reduce the number of candidates of other features. These rules are conservative which means that after applying the three rules the size of an optimal solution is the same as before. A proof can be found in their paper [WWKS01].

Every time a candidate p_i of a feature p is eliminated it is recursively checked whether the rules can be applied in the neighbourhood (i.e., the features of label candidates that conflict with p_i). Since the rules work only on the conflict graph, it is irrelevant of which shape the labels are or for which type of feature (point, line or area) they are applied.

Phase II.

If there are still conflicts in the conflict graph and there are features with more than one label candidate left after applying phase one, phase two heuristically reduces the number of candidates for each feature to at most one. It is a heuristic and thus optimality is no longer guaranteed.

Phase two works as follows: we pick from the list of features that still have more than one label candidate left the one with the most remaining candidates. If there are several features with equal number of candidates it is not specified which one shall be taken and depends on the implementation. Then we delete for this feature the label candidate that has the most conflicts among the candidates. Afterwards we recursively apply the three rules in the neighbourhood of the eliminated candidate. We repeat the whole process in a loop until there are no conflicts any more and all features have at most one label candidate remaining which is assigned to it as its label. The pseudocode of the three rules algorithm with its two phases can be seen in Algorithm 4.1 which is taken from Wagner et al. [WWKS01].

Algorithm 4.1: RULES(features F, candidates C_f for each $f \in F$, conflict graph G)

```
1: //phase I
2: apply the three rules exhaustively to all features
3:
4: //phase II
5: while there are intersecting candidates do
6:    f ← feature with maximum number of candidates in F
7:    delete candidate c of f with maximum number of conflicts in C_f
8:    apply the three rules exhaustively, starting in the neighbourhood of c
9: end while
```

Priorities

The initial version of the algorithm or more precisely of the three rules does not support weighting of label candidates or features. But priorities can be added to the algorithm by restricting the three rules. For that, rule one and rule three have to be restricted so that only those candidates are deleted that have a priority less or equal than the selected candidate (i.e., p_i in Figures 4.1 a and c). Rule two has to be restricted so that only those candidates of p and q are deleted that do not allow a higher value (i.e., most often the sum of the priorities) compared with p_i and q_j (see Figure 4.1b).

4.4 Integer Linear Programming Approaches

Linear Programming (LP) is a mathematical optimization technique to minimize or maximize a linear objective function subject to linear constraints. Formally we can define it as follows.

Let $A \in \mathbb{R}^{m \times n}$ be a matrix and $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ be two vectors of known coefficients.

Then a linear program can be defined as

$$\text{maximize} \quad c^T x \tag{4.3}$$
$$\text{subject to} \quad Ax \leq b, \tag{4.4}$$

where $(c)^T$ is the transpose of matrix c and $x \in \mathbb{R}^n$ is an n-dimensional vector of real variables to be determined. Formula 4.3 represents the objective function (which also can be minimized) and the inequality shown in 4.4 represents the linear constraints which also can be exchanged for $Ax = b$ or $Ax \geq b$. A vector $x \in \mathbb{R}^n$ satisfying constraint 4.4 is called a *feasible solution*. Finding the optimal feasible solution is the goal of linear programming. If no vector x can be found then the linear program is said to be *infeasible* [Zor86, Kla01].

Integer linear programming (ILP) problems are LP problems where all the elements in x have to be integers. A special form of ILP is the *0-1 integer linear programming* (or binary ILP) where all the elements of x are required to be zeros or ones [Zor86]. Both problems, the ILP and the binary ILP, are NP-hard [GJ79], while an LP can be solved in polynomial time [Kar84, GJ79].

A number of algorithms and approaches were developed for the label placement problem using integer linear programming. Zoraster [Zor90] proposed and implemented an algorithm for the label placement problem that uses binary ILP and aims at label number maximization. The algorithm uses a Lagrangean relaxation to produce solutions close to optimal. Ribeiro et al. [RML11] also developed an algorithm based on a very similar binary ILP model for label number maximization. They use a Lagrangean decomposition to partition the problem into small sub-problems.

Consider the following notation: Let i be a feature to be labelled and N the total number of features. Each feature has a number P_i of candidate positions. Each candidate position is represented by a binary variable $x_{i,j}$ where $i \in \{1, ..., N\}$ and $j \in \{1, ..., P_i\}$. If $x_{i,j} = 1$ then the candidate position j is assigned to the feature i as its label, otherwise $x_{i,j} = 0$. Each candidate additionally has a penalty (or profit) represented by $w_{i,j}$. Let further be $S_{i,j}$ the set of index pairs (k, t) of label candidates $x_{k,t}$ with $i \neq k$ that conflict with $x_{i,j}$.

The integer program for label number maximization can be described as follows:

$$\text{minimize} \quad \sum_{i=1}^{N} \sum_{j=1}^{P_i} w_{i,j} x_{i,j} \tag{4.5}$$
$$\text{subject to} \quad \sum_{j=1}^{P_i} x_{i,j} \leq 1, \qquad \forall i = 1, ..., N \tag{4.6}$$
$$x_{i,j} + x_{k,t} \leq 1 \qquad \forall i = 1, ..., N; \forall j = 1, ..., P_i; (k, t) \in S_{i,j} \tag{4.7}$$
$$x_{i,j} \in \{0, 1\} \qquad \forall i = 1, ..., N; \forall j = 1, ..., P_i \tag{4.8}$$

Formula 4.5 shows the objective function that has to be minimized (or maximized if $w_{i,j}$ is seen as profit). Constraint 4.6 shows that each feature can have at most one

candidate assigned as its label, constraint 4.7 describes the overlaps of some candidates and constraint 4.8 denotes that all variables are binary variables. Relating the constraints to the conflict graph, every edge corresponds to one constraint of the form shown in Formula 4.7.

Ribeiro and Lorena [RL06, RL08] additionally investigated a 0-1 ILP model combined with Lagrangean relaxation heuristics for labeling all features and minimize the number of conflicts. In both papers they rely on a conflict graph in the background. The integer program we can use here is similar to the one for label number maximization. But for this Ribeiro and Lorena additionally introduced a new conflict variable $y_{i,j,k,t}$ representing an edge in the conflict graph, i.e., for every $(k,t) \in S_{i,j}$, where $k \in \{1,...,N\} : k > i$ and $t \in \{1,...,P_k\}$ there exists this binary conflict variable. The slightly modified integer program is as follows:

$$\text{minimize} \quad \sum_{i=1}^{N}\sum_{j=1}^{P_i}(w_{i,j}x_{i,j} + \sum_{(k,t)\in S_{i,j}} y_{i,j,k,t}) \tag{4.9}$$

$$\text{subject to} \quad \sum_{j=1}^{P_i} x_{i,j} = 1, \qquad \forall i = 1,...,N \tag{4.10}$$

$$x_{i,j} + x_{k,t} - y_{i,j,k,t} \leq 1 \qquad \forall i = 1,...,N, \quad \forall j = 1,...,P_i, \quad (k,t) \in S_{i,j} \tag{4.11}$$

$$x_{i,j}, y_{i,j,k,t} \in \{0,1\} \qquad \forall i = 1,...,N, \quad \forall j = 1,...,P_i \quad (k,t) \in S_{i,j} \tag{4.12}$$

As can be seen the constraint 4.10 has been strengthened because now exactly one candidate has to be assigned as label to the feature. And in constraint 4.11 now overlapping label candidates $x_{i,j}$ and $x_{k,t}$ can both be 1. But then it follows that the conflict variable $y_{i,j,k,t} = 1$ and this is what we try to avoid. Because by minimizing the objective function shown in Formula 4.9 the conflict variables have to be eliminated or minimized (if elimination is not possible).

By modifying this ILP model, it is relatively easy to adapt this approach to other objectives. Mauri et al. [MRL10] showed a similar strategy combining ILP with a Lagrangean relaxation for the problem of maximizing the number of conflict free labels (MNCFLP). But this objective is not part of this thesis and hence this approach is not further described.

Priorities

In ILP priorities can be added very easily. Because of the penalty factor $w_{i,j}$ that each label candidate has, we simply can use this to express priorities. As mentioned above

for the label number maximization approach we just have to see this factor as a profit factor and then maximize the objective function. For the minimum number of conflicts approach we can not just maximize the objective function since we want to minimize the number of conflicts. But we can multiply its priority with -1, i.e., $w_{i,j} = -1 \cdot p_{i,j}$, where $p_{i,j}$ is the priority of label candidate j of feature i. Hence, the higher the priority the lower its penalty.

CHAPTER 5

User Constraints

Building a framework for a user-centered and semi-automatic map labeling process needs to comprises thoughts on the use cases a real user will perform. It is not sufficient to place as many labels as possible on a map but it strongly depends on the needs of the user (that is the map creator) and on what he or she is interested to show with his or her map. Does one want to create a thematic map like a hiking or political map, a road map for navigation, or does one want to create a city map labeling any kind of point of interest? Is one interested in labeling mountains and lakes or in labeling hotels, churches, bars and sports facilities? Additionally to these considerations it is necessary to know what the expert user wants to do during interaction with the framework to create maps. Does one want to move or delete labels, change their size or define areas without labels?

This chapter will cover these aspects and considerations about constraints given by the user. It first describes the interviews we made to identify which functionality the framework shall include so that it can be used in real world tasks. Then a classification of these constraints is given and lastly the influences to the conflict graph are discussed. Again, as this thesis concentrates on point feature labeling the constraints are specific but not exclusive for point features.

5.1 Interviews

As our thesis concentrates on cartographic map labeling we conducted some interviews with a cartographic expert. The first interview was to identify the tasks a cartographer wants to perform and the second one to review the implemented functionality.

5.1.1 First Interview

In the first interview we presented our basic framework to the cartographic expert to give her an overview of our intended approach. At this point of time the framework already

included the functionality to load label data, filter for specific types of labels and display them on the map. Additionally there already were some algorithms implemented that produced an initial solution (i.e., an initial labeling). Point features not contained in the labeling solution were shown in a different colour and on a mouseover the hidden label candidates of the feature could be shown. A basic implementation of a label enlargement was also existing in the framework representing the task of modifying labels on the map.

The goal of that interview was to get an idea if a cartographer wants to use such a semi-automatic approach and which label modifications he or she uses in every day map creation processes that need to be implemented to guarantee a framework satisfying the experts needs.

After exploring the framework we identified the following list of constraints. It shows the tasks a cartographer mainly performs during map creation.

- Change the font for the label text,
- Change the font size of the label text,
- Apply initial weights (i.e., give a certain type of features a specific priority. E.g., all towns with ten thousand or more inhabitants shall have a higher priority),
- Change the weight of one label,
- Delete irrelevant labels,
- "Stacking" of labels (i.e., multi-line label text),
- Possibility to abbreviate long labels and
- Adding a padding to the label (e.g., add a padding to important labels so that they are prominent on the map).

Besides this listing of tasks a cartographer does during his or her daily map creation process our interviewee told us the usual cartographic label position preferences. In a discrete four position model the ranking of the label candidates would be like it is shown in Figure 5.1.

5.1.2 Second Interview

The second interview was a presentation of our framework and its newly added functionality to the cartographer. The main changes were a number of label modifications and the automatic recalculation of a solution after these modifications. At that moment the following modifications were implemented: change of font size, changing priorities, modification of the label text (i.e., abbreviation and stacking), possibility for padding and deletion of label candidates or whole features.

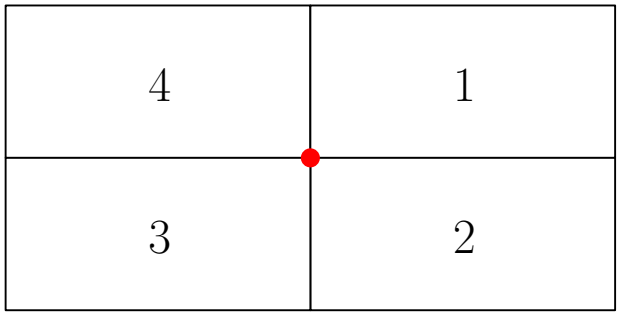


Figure 5.1: Cartographic label position preferences for the discrete four position model.

One point she mentioned in the second interview was the storage of the label text modifications. In our framework the label text (i.e., the label meta data) is directly modified when the user uses abbreviation or stacking. She told us that normally the label text remains unchanged and only the operations like stacking or abbreviations are stored in an extra data field and are then always applied to the label text when it is displayed. But besides this she was satisfied with our implementation and wishes that the tools she currently works with would already support parts of our framework to enhance her map creation process.

5.2 Label Modifications in our Framework

Starting from the input of the interview with the cartographic expert we decided to implement the following slightly different list of label modifications. The reason why we did not implement the list of the expert one-to-one is that the implementation of some of the label modifications would exceed the realm of this thesis and would not add any additional value in the sense of our thesis interests.

1. Change the font size of a label,
2. Change the weight of a label candidate,
3. Add a padding to the label,
4. Abbreviation of the label text,
5. Stacking of the label text,
6. Text modification,
7. Delete label candidate,
8. Delete a point feature (i.e., deletes all its label candidates) and
9. Fixate a label candidate (i.e., set it as unchangeable label for the feature).

	Modification	Effect
1	Font Size	$A \oplus B$
2	Weight	–
3	Padding	$A \oplus B$
4	Abbreviation	$A \oplus B$
5	Stacking	$A \vee B$
6	Text Modification	$A \oplus B$
7	Delete Label Candidate	B
8	Delete Point Feature	B
9	Fixate Label Candidate	B

Table 5.1: An overview of the implemented label modifications and their effect on the conflict graph.

Compared to the list from interview one we left out the task of changing the font for the label text since it does not add any new type of modification to the framework. Changing the font is like changing the font size just a resizing of the label itself. Thus we decided to leave it out for our thesis. We also omitted the initial weights for a specific type of features as it works the same way as changing the weight for one specific label.

Instead of implementing complex automatic abbreviation and stacking techniques we decided to let the user do this task manually by simply editing the label text. This is why we have the additional modification number six in the enumeration above where the user can edit the whole label text. The additional possibility to fixate a label candidate is more or less a deletion of all label candidates that conflict the fixed one.

5.3 Classification of the Constraints

To include these constraints in our framework we had to classify them in respect to the conflict graph we use. There are two possible operations on the conflict graph:

(A) Add an edge or

(B) Remove an edge.

Looking on the list provided in Section 5.2 it can be seen that most of the modifications result in a resize (i.e., enlargement or downsizing) of the label respectively its boundings. Modification one and modifications three to six are of this kind. An enlargement of a label can effect new overlaps between labels and thus new conflicts, i.e., new edges have to be inserted into the conflict graph. On the other hand downsizing a label can effect that labels do not overlap and thus do not conflict any more, i.e., edges can be removed from the conflict graph.

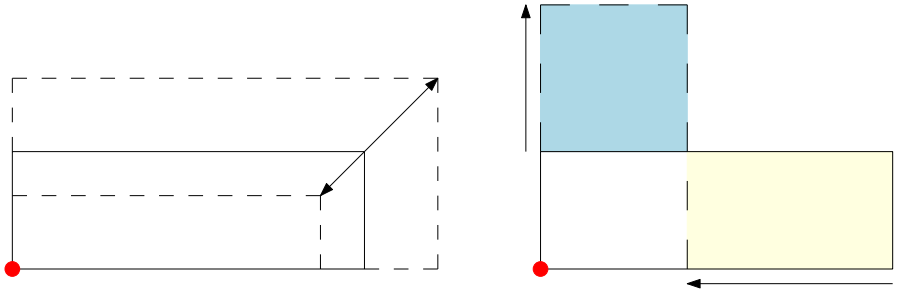


Figure 5.2: Uniform resizing of a label on the left, non-uniform resizing on the right.

The same applies to modifications seven to nine. Deleting a label candidate or a point feature is just a remove of the corresponding node and all its edges from the conflict graph. The fixation of a label is nearly the same, but without deleting the node from the conflict graph. Instead the nodes of the conflicting label candidates are deleted. Since the node of the fixated label remains without any conflicting edge it can always be selected by any labeling algorithm.

The only remaining label modification number two can not be represented in the conflict graph because a change of the priority does not change the size of the label and thus does not add or remove any edges from the conflict graph. This modification has to be considered in another way from the labeling algorithms. Table 5.1 shows an overview of all implemented label modifications and their effect to the conflict graph. Effect A means that possibly edges have to be added and B means that possibly edges have to be deleted (the letters correspond to the enumeration above). The $\oplus$-operator indicates that for most of the modifications only one effect to the conflict graph can occur. Either the label grows in both directions or the label shrinks in both directions and hence either new edges have to be added or edges have to be deleted, see the left illustration in Figure 5.2. Note that the stacking modification can (but does not have to) effect both at the same time because it is the only modification where the sides of the label can be extended in one direction while the sides are shortened in the other direction, see the right illustration in Figure 5.2. So previous conflicts of the yellow area are deleted and new conflicts of the blue area are added.

Considering the efficiency of the two constraint classes, both are fast and have small effort to update the conflict graph. Adding or removing a single edge from the conflict graph can be done in constant time. A node lookup in the conflict graph can also be done in constant time in a good implementation. Assuming the worst case that a node in a conflict graph with n nodes is connected to every other node and we want to delete every edge (that are $n-1$ edges) it can be done in $O(n)$. The worst case of adding $n-1$ edges is equal to that. So all label modifications can at least be done in $O(n)$.

5.4 Optimal Solution after Label Modification

Considering an existing optimal solution (i.e., the current labelling) then all above mentioned modifications induce a recalculation of this solution because after updating the conflict graph it may be invalid or not optimal any more. The special case of constraint two (changing label weights) is excluded in the following since it does not affect the conflict graph. The handling of this modification type depends on the applied algorithm.

Since the label number maximization is the main objective for label placement in this thesis we focus on this kind of objective in the following remarks. Hence, when we speak of an optimal solution it means a solution where a maximum number of features is labelled or in other words a maximum independent set in the corresponding conflict graph. To show the influence of the constraint classes on the size of optimal labeling solutions we formulate the following two theorems.

Theorem 1. *Let S be an optimal solution for a labeling instance containing k labels. Then by removing one edge from the corresponding conflict graph a newly calculated solution S' contains at most $k+1$ labels.*

Proof. Assume to the contrary that the newly calculated solution S' contains at least $k+2$ labels. Since only one edge connecting two nodes in the conflict graph was removed and the remaining conflict graph stayed untouched, our assumption implies that both nodes are added to the new solution S'. If we again add the edge between these two nodes, only one node has to be removed from solution S' to maintain a maximum independent set and we have found a valid solution with $k+1$ labels. Hence the initial solution S could not have been maximal and we have found a contradiction. □

Theorem 2. *Let S be an optimal solution for a labeling instance containing k labels. Then by adding one edge to the correspondent conflict graph a newly calculated solution S' contains at least $k-1$ labels.*

Proof. There are three cases to be considered.

- Case 1. None of the nodes incident to the added edge were contained in the initial solution S. Then adding the edge between them does not affect the solution and hence $S' = S$ with $|S'| = k$ which satisfies the requirement of at least $k-1$ labels.
- Case 2. Only one of the two nodes that the edge connects was contained in the initial solution S. Also in this case, the added edge does not influence solution S and remains valid.
- Case 3. Both ends of the added edge are in the initial solution. By deleting one of the two ends from our solution S we get a new valid solution S' with $k-1$ labels. In this case S' might not be maximal but it is a valid solution.

□

CHAPTER 6

Development and Implementation of the Prototype

One aim of this thesis was to develop a prototype of a simple user-centered map creation tool that supports a cartographic domain expert user in his or her map creation process. It should provide different automatic label placement algorithms that produce an initial labeling solution which is then refined and improved by the user. The framework should be able to handle the user constraints identified in Chapter 5 and the algorithms in the background shall respect them to recalculate an optimal labeling and optimize the previous solution.

The prototype we present in this chapter concentrates on the point feature label placement on static geographical maps with a discrete label positioning model and axis-parallel rectangular labels. For this first attempt we leave aside the line and area feature labeling and other positioning models because this would go beyond the scope of this thesis.

This chapter describes the basic approach of the framework (Section 6.1), gives an overview of the used technology (Section 6.2) and describes some relevant implementation details in Section 6.3, e.g., about the main data structures or the input data. Section 6.4 explains the user interface and its functionality and lastly in Section 6.5 all the labeling algorithms available in the application are described and compared.

6.1 Basic Workflow of the Framework

The basic workflow of the framework can be described by the following steps. Starting from a non-labeled map the user can import a set of data points that he or she wants to be labeled on the map. By panning around in the map area of the framework he or she can select the region he or she wants to label and by zooming in and out one can or more precisely has to set the zoom level. Since we investigate the point feature label

placement on static maps the zoom level has to be fixed. In contrast to the fixed zoom, panning around the map is possible even if a solution was already calculated since it has no influence to an existing solution.

After importing a file the framework parses the input data and generates suitable label candidates for each point feature dependent to the position model, i.e., the fixed four-position model in our implementation. The spatial information of the features and their label candidates are stored in a quadtree which is then used to generate a conflict graph for storing the conflicts (i.e. the overlappings) between the candidates.

Based on this the user can now filter for specific types of features or apply different algorithms and display the calculated labeling. Additionally he or she has the possibility to modify, delete or fixate single label candidates. After a label modification the framework automatically recalculates a new valid solution. The user can again specify which algorithm is used for the recalculations. Since this step is possibly done very often the recalculations shall be reasonably fast.

6.2 Used Technology

For the implementation of our prototype we used *Java 8* as the programming language. Since online maps are quite common and popular in these days, there exist a bunch of *JavaScript* libraries for interactive maps. Hence, as the interactive map is a central point in our application a web-based approach seemed to be a good starting point for our application. For this we use the lightweighted *Play Framework* (version 2.6) in combination with *Leaflet* (version 1.0.3) and *D3* (version 4.9.1) for displaying a map and drawing labels.

To give an idea of the developed prototype the basic user interface is shown in Figure 6.1. It shows the large map area where the user can pan and zoom around and can see the current labeling. Most of the action elements, i.e., buttons and dropdowns for loading a file, filtering or applying an algorithm are in the left sidebar.

As the created tool is web-based it can basically be run on any browser and on any operating system, but we mainly tested it with the *Google Chrome* browser (version 66.0) on a 64-bit *Windows 10* operating system. Test runs on a Linux system showed that the label drawing does not work totally correct despite installing the same Java version and libraries. The application is not designed to be used on mobile devices or devices with small screens The tool is tested on and optimized for a 21 inches monitor with resolution of 1920x1080.

6.2.1 Play Framework

The *Play Framework*[1] is an open source web application framework and is available for building scalable web applications with Java and Scala. It is based on a lightweight,

[1] `https://www.playframework.com/` version 2.6

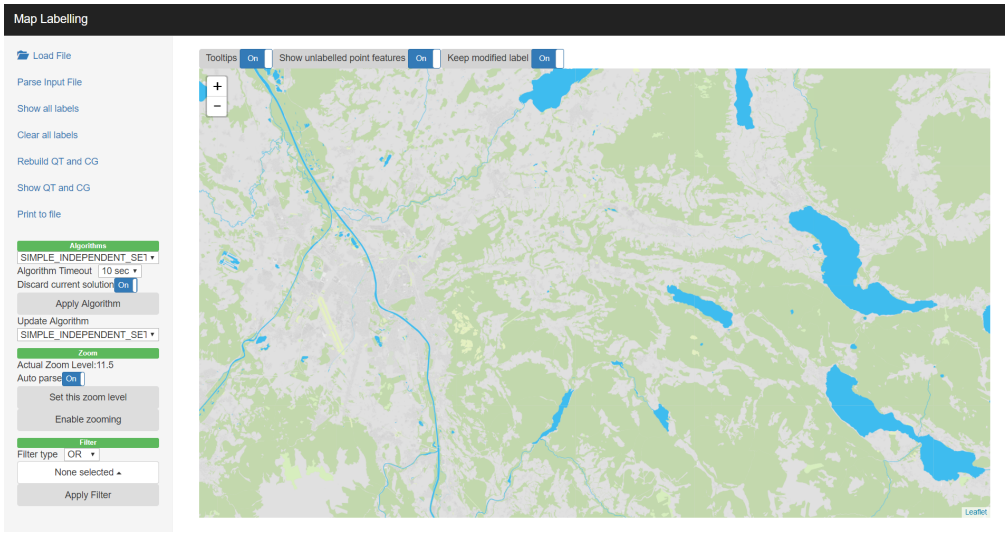


Figure 6.1: The basic user interface of the developed prototype.

stateless and non-blocking architecture. It provides hot code reloading, displays errors directly in the browser and supports REST and WebSockets. The latest version is 2.6 which by default has *Akka HTTP*[2] as its server backend. It further uses *sbt*[3] as the build tool and for dependency management.

6.2.2 Leaflet

Leaflet[4] is an open-source JavaScript library for building web mapping applications. It was initially developed by Vladimir Agafonkin and first released in 2011. The library supports most mobile and desktop platforms and supports the current web standards HTML5 and CSS3. Besides OpenLayers and the Google Maps API it is one of the most popular JavaScript mapping library [Bac]. It focuses on simplicity, performance and usability and thus the core has only basic functionality which is sufficient for our needs and for most real-life use cases. The small basis can be extended with a large amount of plugins. It has a well-documented and easy to use API and hence allows less experienced users or users without GIS background an easy creation of interactive maps.

For the interactive map Leaflet uses a tiled map. The approach of *tiled maps* is currently the most popular way to display and navigate online maps. Instead of loading the world map as a whole image, the map is sliced into square images of equal size and then showed in an seamless grid arrangement. These images that are most often 256x256 pixel wide, are called *tiles*. The advantage of tiled maps is, that only those tiles have to be requested and downloaded that are currently needed for the specific zoom level and map area that

[2] `https://doc.akka.io/docs/akka-http/current/index.html` version 10.1
[3] `https://www.scala-sbt.org/` version 0.13
[4] `https://leafletjs.com/` version 1.0.3

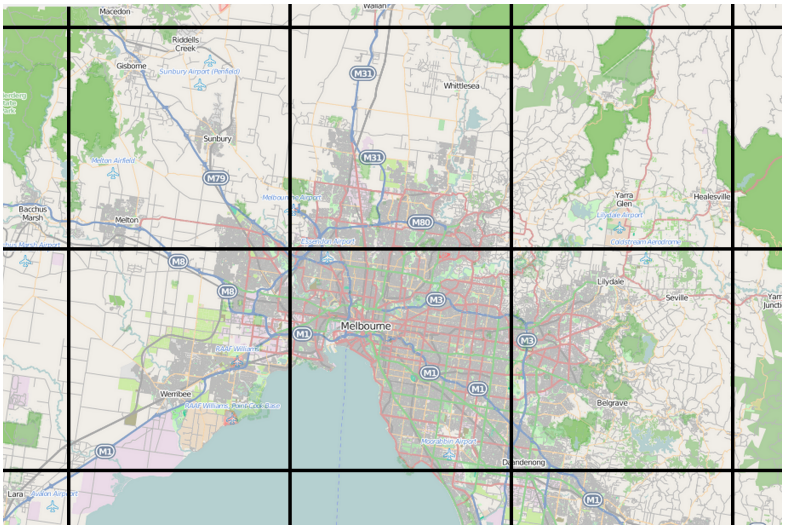


Figure 6.2: A tiled web map.[5]

is displayed. Additionally when a user pans around the map, most of the tiles are still relevant and can be kept and only the missing tiles have to be fetched. This improves the user experience and the performance of online maps since the tiles can be precomputed on the server instead of rendering them in the browser ever and ever again. For the user there is no visual difference since the tiles are seamlessly joined to a larger image and the map looks like one.

At the outer most zoom level the entire world can be displayed in a single tile. The tiles can often be fetched through a web server with a URL like `http://.../Z/X/Y.png` where Z is the zoom level and X and Y identify the tile. Figure 6.2 shows an example of a tiled web map. Normally the tiles are displayed without gap.

Leaflet natively supports different layer types such as GeoJSON, vector or tile layers. Other types may be supported by a plugin. Like other web map libraries Leaflet displays one basemap and several possibly interactive overlays, e.g., for markers, popups or tooltips.

A Leaflet map can be added to a web page with the simple JavaScript code snippet shown in Listing 6.1. A map element is bound to an HTML element such as a `div` (with ID `mapID`) and the viewpoint and zoom level are set (line one). Other layers can then be added to the map element like it is shown in line three. In this example a tile layer is added to the map. Leaflet can be used with different tile sets by just exchanging the URL to the tile web server. In our thesis we use a tile set with the name *Hydda* from the Swedish OSM team [swe]. The "base" tile set has no labels which satisfies our requirements. The Leaflet library is accessed through the variable L. Adding a label (or

[5] `https://en.wikipedia.org/wiki/Tiled_web_map`

as it is called in Leaflet, a *marker*) at a specific location, is as simple as it is shown in line five.

```
1 var map = L.map('mapID').setView([51.505, -0.09], 13);
2
3 L.tileLayer('http://{s}.tile.openstreetmap.se/
4         hydda/base/{z}/{x}/{y}.png').addTo(map);
5 L.marker([50.5, 30.5]).addTo(map);
```

Listing 6.1: Creating a map with a tile layer in Leaflet.

6.2.3 D3 Library

Our labeling problem instances require to draw a lot of labels respectively markers. Hence, the usability and performance of the application depends on the efficiency of label rendering. It exposed, that the Leaflet markers are quite slow when creating and drawing a large amount of them. Panning around was almost impossible on a map with a hundred labels. We decided to use Leaflet just for the map, i.e., for map control (zooming and panning) and for providing the tile set, and use another faster library for drawing the labels.

D3 (*Data-Driven Documents* or *D3.js*)[6] is a JavaScript library for dynamic data visualization using web standards like SVG, HTML and CSS. It combines visualization and interaction techniques with a data-driven approach to Document Object Model (DOM) manipulation. Arbitrary data can be bound to a DOM and then data-driven transformations can be applied to the document.

In our thesis we use D3 to bind the label data to a DOM element and use it to generate SVG elements for every label candidate. More precisely we generate for each label candidate a rectangle indicating the boundings of the label, a circle representing the point feature it belongs to, and a text element describing the point feature. Since Leaflet supports SVG layers, the resulting label representations created by the D3 library can simply be added to the Leaflet map. It turned out that this approach is much faster and we are able to display up to 5.000 labels without slowing the application.

6.3 Implementation Details

In this section we give some details about the implementation of our prototype, especially about the implementation of the main data structures and the input data.

[6] `https://d3js.org/` version 5.5

6.3.1 Basic Architecture

Our application is a classical client-server web application. This means that the logical calculations run on the server and the graphical user interface is rendered in a browser on the client. They both communicate with each other via web sockets.

One aspect about our application is, that we do not use a database to store the labeling data. Everything that we need is kept in the internal memory. The advantage of that is that we did not need to think about storing our large graphs in suitable databases and implement efficient database connectivity and database queries. Of course the disadvantage is that after our application shuts down the labeling is lost. But for the use cases of our prototype version it is sufficient to keep the data in memory. It can be changed in future work.

6.3.2 Client-Server Communication

As mentioned client and server communicate with each other over web sockets. *WebSocket* is a communication protocol that provides full-duplex communication over TCP, i.e., communication is allowed in both directions simultaneously. So as long as the connection is kept open, both can send messages and even if there was no client request the server can send content to the client.

Client and server communicate with each other by sending their data in a special *JSON*-format. JSON stands for JavaScript Object Notation and is a lightweight data-interchange format. Since it is just text it is language independent. The communication structure of our prototype is specified by

```
{"type" : SOME_TYPE, "data" : SOME_DATA}
```

where `type` specifies the type of the message and `data` the data that are transmitted. When the client sends a message to the server the value of `type` is one of the types defined in `ClientMessageType` which is an enumeration of all message types the server can understand. And when the server sends a message to the client it is one of the types defined in `ServerMessageType` which enumerates all types the client can understand. The value of the `data` element contains the data the client or the server wants to transmit. It is a string in JSON format and depends on the message type.

6.3.3 Input Data

To draw labels on the map we first need to get input data. Since we want to create a realistic usable framework, we also want to use real geographical data which we can get from *OpenStreetMap* (OSM). OpenStreetMap [osm] is a collaborative project collecting geospatial data (e.g., ways, buildings, places) and making them available to everyone (Open Data). The data are made available under the Open Database License (ODbL) [odb].

There are several locations where we can get the data. The most famous snapshot of the OSM database is *Planet.osm* [pla]. It is regularly updated and contains all nodes (i.e., points in space), ways (i.e., linear features and area boundings) and relations (i.e., relationship between nodes, ways or other relations) of the whole map, i.e., it is a complete copy of all OSM data. The data can be fetched as compressed XML file or in PBF (Protocolbuffer Binary Format) which is much smaller than the zipped XML format. The current size of the zipped XML file is about 66 GB. This is way to much data for our application. Fortunately there are many mirrors where we can also get excerpts of specific areas (e.g., individual countries) of the whole data set. We use the German provider *Geofabrik* [geo] for downloading.

Although PBF is smaller in its size and faster to read we can not directly use it because of its binary nature. It is easier for us to use the XML file. Actually the OSM data contain far too much and for our purpose irrelevant information (e.g., ways and relations). Parsing it directly would be too time consuming. We use a small piece of code to extract the information we need for our application from the large XML files. The resulting file is in JSON format and is far smaller. For example the XML data of Iceland currently has about 714.000 kilobytes whereas the extracted JSON file has only about 500 kilobytes. The structure of the file is an array of JSON Objects. Every object has the keys “label” (the name of the feature), “mc” (the master category), “sc” (a second category) and the coordinates “x” and “y” of the feature in *WGS 84 Web Mercator* (or EPSG:3857) [wgs] projection which is popular for web mapping applications. Listing 6.2 shows an excerpt of the Iceland dataset.

```
[
{"label": "Reykjavík","mc": "place","sc": "city",
        "x": -2442598.6163094793, y": 9386931.633935148},
{"label": "Dalvík","mc": "place","sc": "town",
        "x": -2062799.011391919,"y": 9868939.164436119},
{"label": "Ólafsvík","mc": "place","sc": "town",
        "x": -2639210.8560456797,"y": 9581012.359032592},
{"label": "Akranes","mc": "place","sc": "town",
        "x": -2458307.410593027,"y": 9430743.98638051},
{"label": "Húsavík","mc": "place","sc": "town",
        "x": -1930395.7648896866,"y": 9888711.321015112}
]
```

Listing 6.2: JSON input format for our application.

Even though we create our own input files and input structure, they are somewhat specific to the OSM data because we use the OSM specific category structure (master and second category). However, the input is not tied to OSM data since data from other sources may can also be translated to the two category structure.

A problem we faced during the implementation of the prototype was to handle different coordinate systems that are used by the tools we use. Besides translations between the different world coordinate systems WSG 84 (used by Leaflet) and EPSG:3857 (data from

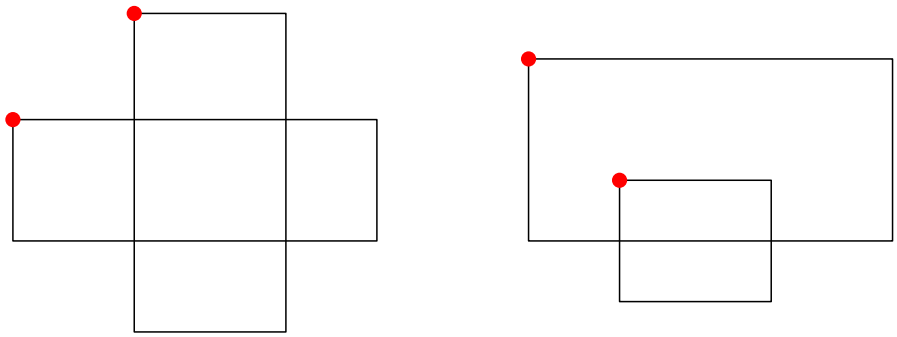


Figure 6.3: Special label conflicts that could not be detected from both (left) or only detected from one side (right) in a range query.

OSM) we also had to attend the flip of the y-coordinates. While the world coordinates have their origin in the lower left corner and Y-values grow upward like it is in ordinary cartesian coordinate systems, we have to draw the labels originated in the upper left corner and with Y-values growing downwards like it is in many graphic libraries. It was a source of inconsistencies between conflicts in the conflict graph and visible conflicts in our labeling visualization that we did not find at first glance.

6.3.4 Implementation of the Data Structures

Quadtree

For the storage of the spatial data our framework uses an implementation of a point region quadtree. As mentioned in Chapter 3 a compressed quadtree is more memory efficient or there are other better performing alternatives but the point region quadtree is sufficient for the purpose of this thesis. The quadtree is implemented as an interface and therefore can be replaced by a more efficient quadtree easily.

In our application it is required to store all the label candidates in the quadtree preserving their location and extension bacause the quadtree is used to determine overlaps. Storing only the point features was not sufficient for detecting conflicts. Each label candidate has an anchor point (i.e., the upper left corner of the label) which is stored in the quadtree. Our version of the point region quadtree has a capacity of one point per quad. So in every leaf there can be at most one label candidate stored. The inner nodes do not store any point data.

With this approach we were able to check if the anchor point lies within an window or within another label, but we could not detect special cases like they are shown in Figure 6.3. Assuming in this illustration the label boundings of one label as the window in a range query, then the left conflict case could be detected by none of the two labels and the right one just from the larger one, since it contains the anchor point of the smaller label.

Hence we modified the point region quadtree so that each inner node now contains a list of labels that intersect the quad node. On a range query we also check each quad that intersects the window if it contains a label that intersects the window. This approach helps and guarantees us to find all conflicts between the labels. One can easily see that this implementation is not the most efficient one, but searching and implementing a more optimal possibility for storing and detecting label conflicts is not part of the thesis and can be done as future work.

Conflict Graph

The conflict graph of our application is implemented as a set of vertices where each vertex represents a label candidate, as it is described in Section 3.2. A vertex stores the corresponding label, the set of edges and the set of deleted edges. The vertex has an edge to another vertex if the label candidates they represent are in conflict. The set of deleted edges is stored so that we can get the original conflict set after some deletions. The conflict graph is implemented as an interface so that it can be easily exchanged by another implementation.

We tried different ways to store the vertices but the best performing was to store them in a `HashMap` with the ID of the label as a key and the Vertex as the value. Searching for a vertex in the hash map can be done in constant time and hence adding an edge respectively a conflict between two vertices can be done in $O(1)$. Removing an edge or checking if a label conflicts with another is in $O(k)$ where k is the number of conflicts of the (first) label, because on these operations we have to run through all edges of the corresponding vertex to find the correct one. Imagine the worst case of one large label candidate where all other label candidates (except its siblings) lie within or intersect this candidate. Then it has $n - 1$ conflicts, where n is the number of total label candidates. Hence we can say that a removing an edge or checking for a conflict is in worst case in $O(n)$ although this case is very unlikely because normally $k \ll n$.

An expensive operation in our conflict graph implementation is to receive the full set of conflicts because for this operation we have to run through the whole set of vertices and through all edges of the vertices to retrieve the whole set. Hence we have a complexity of $O(n + m)$ where n is the number of vertices and m the number of conflicts in the conflict graph. As mentioned in Section 3.2, in worst case the number of edges is quadratic in the number of features ($m = O(n^2)$). So the complexity of this operation is in worst case in $O(n^2)$.

6.3.5 Position Model

For our axis-parallel labeling approach we implemented two position models in our prototype. The fixed four-position model and the fixed eight-position model, as described in Section 2.1. But we did not implement the possibility for a user to switch between the models. Our investigations concentrate on the fixed four-position model and hence the application uses this model. The eight-position model was only implemented for testing

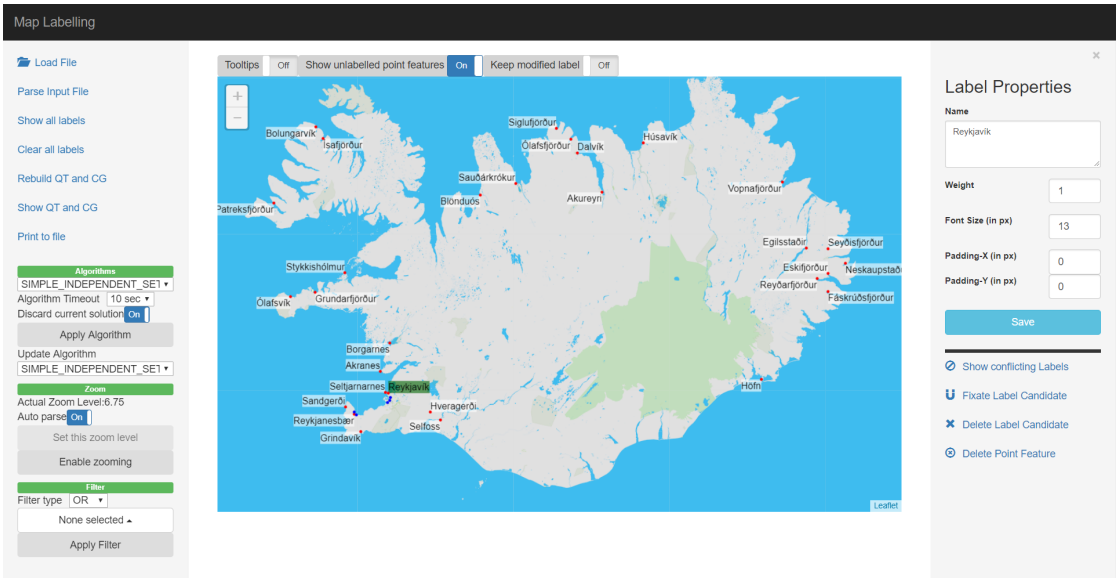


Figure 6.4: User interface showing a labeling of Iceland and all sidebars.

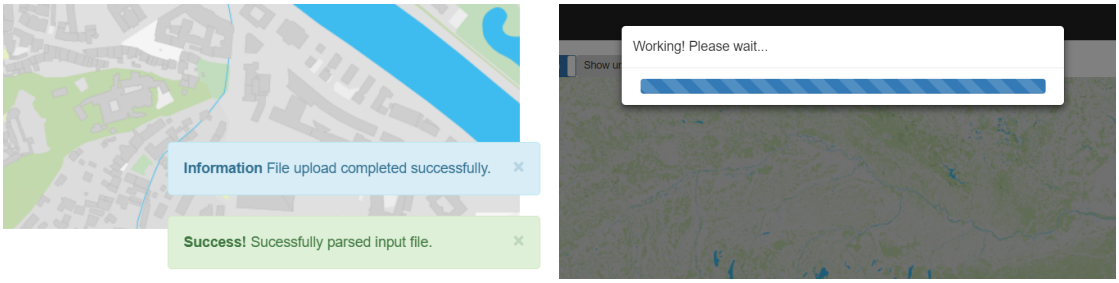
reasons. Nevertheless a model switch can be added in the future. Changing the position model would need to reparse the input data. Although we only use one position model, our framework is designed in a way so that the backend can handle any kind of discrete position model without modifying our implementation.

6.4 User Interface

In this section we want to describe the user interface and its functionality. For our thesis the design of the interface was not in the foreground but to satisfy the functional requirements.

Our application is a one site application. The basic user interface was already shown in Figure 6.1. It consists of the large map area as its dominant UI part, where the user mostly works on. Here the current labeling is shown and the user can edit specific labels. On the left hand side there is a sidebar containing the main functional action buttons, e.g., for loading a file or filtering the labels (see Section 6.4.1). On top of the map area there are three minor options. The first one toggles tooltips (i.e., it toggles if tooltips are shown for a label when the mouse cursor is moved over it), one for toggling unlabelled features (i.e., it toggles if unlabelled point features are drawn on the map) and one specifying if a label shall be kept after a label modification was applied.

Figure 6.4 shows a labeling of cities and towns of Iceland in our application. The point features are displayed as filled red circles. The labels are drawn as white rectangles with a black text identifying the corresponding point feature. Those point features that have no label assigned (because they were deleted or they could not be labeled because of their



(a) (b)

Figure 6.5: Feedback of the application shown to the user. Small message boxes (a) to inform the user and blocking the user interface on long running operations (b).

conflicts with other point features) are shown with a filled blue circle only. As mentioned above the unlabelled point features can also be hidden. Some hidden point features can be seen in the neighbourhood of Icelands capital Reykjavík, i.e., the green label in the figure.

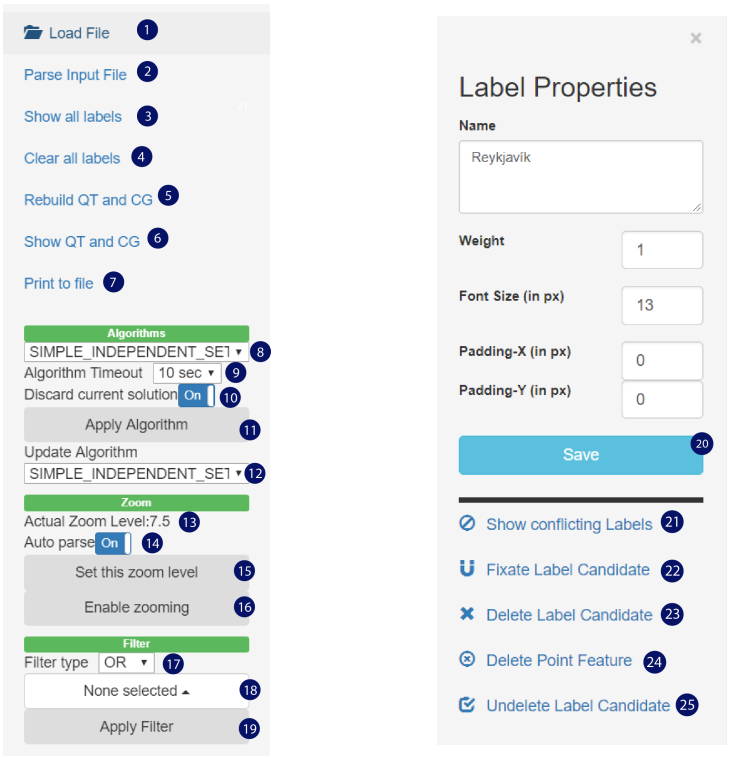
By clicking on a label its background colour changes to green and a label modification area on the right side of the page pops up. In the label modification area the user can apply the different modifications to the label candidate or its point feature (see Section 6.4.1).

Concerning user feedback the application informs the user via small message boxes on the lower right corner, e.g., after successfully loading a file or on any kind of error (see Figure 6.5a). Since our application has some long running operations and calculations the user interface is blocked on these operations and a waiting progress modal is shown as can be seen in Figure 6.5b .

6.4.1 Sidebars

In the following the numbers in brackets refer to Figure 6.6a and Figure 6.6b respectively. Figure 6.6a shows the left sidebar of the user interface. “Load File” (1) loads the input data from a file which then can be parsed (2) to create the label candidates as well as the quadtree and the conflict graph. All available label candidates can be shown (3) or hidden (4) in the map. The quadtree and the conflict graph can be rebuilt (5) dependent to the current solution. “Show QT and CG” (6) is a debug function to display a visual representation of the current quadtree and conflict graph. The current labeling can also be printed to a file (7), i.e., all point features except the deleted ones. The file format is the same as for the input data (see Section 6.3.3) and hence attributes like weight, font size or fixations can not be stored. The label candidates are generated when the file is read in again. We may switch to a more sophisticated file structure storing label meta data in the future.

In the Algorithm area the timeout dropdown (9) defines the timeout after which the



(a) (b)

Figure 6.6: The left (a) and the right (b) sidebar.

algorithm selected in (8) is aborted. The discard option (10) defines if the current solution shall be discarded when applying an algorithm. A click on the apply button (11) applies the algorithm selected in the algorithm dropdown (8). Additionally an update algorithm (12) can be selected that shall be used for the recalculations after a label modification.

The zoom area shows the current zoom level of the map (13). This does not need to be the same as the zoom level set on the server. The auto parse option (14) defines if the file shall be automatically reparsed when setting a new zoom level. By clicking button "Set this zoom level" (15) the zoom level on the server is set to the current zoom level of the map (13). This disables zooming in the map area. The input data have to be reparsed. Zooming can be switched back on (16).

The filter type (17) defines if the categories selected in the filter selection (18) shall be combined with an "OR" (i.e., a label in the filtered set has to have at least one of the specified categories) or an "AND" (i.e., a label has to have all the specified categories). The filter selection (18) is a multiselect dropdown to select some categories to be filtered. The filter can be applied by clicking the apply button (19).

Figure 6.6b shows the right sidebar (or label modification area). It consists of an area where properties of the label are displayed and an area with buttons for the label

modification. The following label properties are shown:

- Name: The text of the label candidate.
- Weight: An integer value representing the weight of the label candidate.
- Font Size: The font size of the label text in pixel.
- Padding: Vertical and horizontal padding of the label candidate in pixel. This moves the label candidate away from its corresponding point feature.

Changes to the properties can be saved by clicking the save button (20). All label candidates that are in conflict with the currently selected label candidate can be marked in the map area (21). The label candidate can be fixated (22), i.e, all conflicting label candidates are deleted. Several fixated labels are allowed to overlap and have to be solved manually by the user. The selected label candidate (23) or the corresponding point feature (24) can be deleted. Deleting a point feature simply deletes all its label candidates. A label candidate can be undeleted (25). This option is only visible for deleted label candidates.

A label modification can either be applied to a single label candidate or to all label candidates of a point feature. Modifications that change a single label candidate are changing the weight and fixating or deleting the label candidate. All other modifications are always applied to all candidates of the point feature.

Referring to the list of label modifications in Section 5.2 which we defined through to the interviews with the cartographic expert, we can apply these modifications with the following user interface actions. Font size, weight and padding each have separate fields where we can change them. All text modifications (including stacking and abbreviations) can be done in the text field for the label name. Clicking the save button then applies the modifications. Deleting label candidates or point features and fixating a label candidate each have separate action buttons.

6.4.2 Visual Appearance of the Labels

The visual appearance of the label shows the current state of the label. Table 6.1 lists all possibilities. Some states allow combinations of effects, i.e., we can combine a border colour with a background colour. For example if a deleted label is selected the visual appearance changes to a green background with a red border. The light blue background from the hidden label candidates can not be combined with any other visual effect. Additionally if we move the mouse over a deleted or fixed label the border colour does not change. The only difference is then the bold text.

Appearance	Occurs	Explanation
white background, no border	label candidate is shown	This is the appearance of any "normal" label candidate.
green background	click on label candidate	The label candidate is currently selected. The modification area shows the properties of this label candidate. See Figure 6.7a.
light blue background	click on hidden point feature	The point feature and hence all its label candidates are hidden. See Figure 6.7b.
red background and red dashed border	click on "show conflicting labels" in the modification area	The label candidate is in conflict with the selected label candidate. The red dashed border line shows the border of the label candidates including their paddings. See Figure 6.7c.
blue border	label candidate is shown	The label candidate is fixated and all of its conflicting labels are deleted. See Figure 6.7b.
red border	label candidate is shown	The label candidate is deleted. See Figure 6.7d.
black border and bold text	on mouseover	If tooltips are enabled then a box with infos about the label candidate beneath the mouse cursor is displayed. Otherwise its just an indicator which label would be selected on a mouse click. See Figure 6.7e

Table 6.1: Description of all the visual appearances of a label candidate and its meaning in our application.

Figure 6.7: Different visual appearances of the label candidates.

6.5 Implemented Algorithms

This section describes the algorithms we implemented in our framework. Subsection 6.5.7 shows an overview and comparison between them.

6.5.1 `ILP`-Algorithm

Our *`ILP`-Algorithm* (Integer Linear Programming Algorithm) is an implementation based on integer linear programming presented in Section 4.4. There we described two different objective functions for the algorithm. Our implementation tries to minimize the number of conflicts between the labels (i.e., we implemented the objective function shown in 4.9 and Constraints 4.10 to 4.12). By simply changing the objective function and constraints we could transform this algorithm to a label number maximization algorithm.

For solving the 0-1 ILP the algorithm uses the *Gurobi Optimizer*[7]. Gurobi is a software for mathematical optimization and solving linear programms. So the steps of our algorithm are as follows: First we create a Gurobi model (i.e., a 0-1 integer linear program) from our labeling instance, then we simply pass it to the Gurobi optimizer and then we translate the solution back to a solution for our labeling problem. The `ILP`-Algorithm supports weights by nature of linear programs, but it does not support the possibility to keep a previous calculated solution.

6.5.2 `MaxHS`-Algorithm

The *`MaxHS`-Algorithm* is an algorithm based on SAT or more precisely on PMAX-SAT. The foundations for this algorithm can be found in Section 4.2. The algorithm first formulates our current labeling instance as weighted PMAX-SAT instance and writes it to a file, since most solvers are designed to read the input from a file. On this file an external MAX-SAT solver is called. The algorithm then reads the output of the solver that contains an interpretation for the SAT instance passed to the solver and translates it back to our labeling. The solution calculated by the SAT solver is exact, i.e., if the algorithm returns a solution, it is guaranteed, that the sum of weights among all possible other solutions is maximal. If the label candidates all have the same weight then the number of labels in the solution is also guaranteed to be maximal.

Besides weighting of label candidates this algorithm also supports the possibility to keep an initial solution as much as possible. This is done by adapting the weights of the labels contained in the solution as it is described in Section 4.2.

To choose a solver for our weighted PMAX-SAT formulation the International Conference on Theory and Applications of Satisfiability Testing (SAT) in Melbourn, Australia [sat] was a good starting point. One part of this conference is the MAX-SAT evaluation where different MAX-SAT solver are tested against each other. It showed that the solver called *MaxHS*[8] performed best for weighted instances. See the evaluation results in [max].

[7] `http://www.gurobi.com/` version 7.5
[8] `http://www.maxhs.org/` version 3.0

Since it also depends a bit on the input set, the solvers may perform differently for certain input sets. So we compared them on our specific data sets but came to the same result, that the MaxHS solver is the best choice for us.

The MaxHS solver was developed by Jessica Davies and Fahiem Bacchus [DB11]. It is a solver for different MAX-SAT problems and takes its input of hard and soft clauses in WDIMACS format. An additional benefit of MaxHS is that it supports floating point weights unlike many other MAX-SAT solvers that require integer weights.

6.5.3 `MNC`-Algorithm

The *`MNC`-Algorithm* (Minimum Number of Conflicts Algorithm) is an algorithm that labels all features while trying to minimize the number of conflicts between the labels. This algorithm is an implementation of the approach presented by Cravo et al. [CRL08]. It uses the greedy randomized adaptive search procedure (GRASP) which is an iterative process to find heuristic solutions. Each iteration consists of a construction phase and a local search phase. In the first phase a feasible solution is constructed which is then improved in a heuristic local search phase to find a local optimum. The best solution over all iterations is kept. Cravo et al. describe that they achieve best solutions when allowing 100 iterations, but for our instances so many iterations are not required, because an acceptable solution is found within the first few iterations. So we restrict the iterations for performance reasons to ten.

The `MNC`-Algorithm supports weights by nature but as presented in their paper they are seen as a penalty. So we multiply them with a factor -1 so that a higher weight leads to a lower penalty and hence the label candidate is more likely to be selected. Despite the objective of minimizing the number of conflicts the algorithm also maximizes the sum of weights of the selected label candidates. The possibility to keep an initial solution and modify it as little as possible is currently not implemented for this algorithm.

6.5.4 `Simple`-Algorithm

The *`Simple`-Algorithm* is an algorithm we implemented for our first fast tests. It is a greedy, fast and straight forward approach with no optimization, which just runs through the list of all label candidates sorted as they were added to the list, selects one and adds all label candidates that are in conflict with it to a "blocked" set to indicate that they can not be selected any more. For the blocked set we use a `HashSet` which stores the IDs of the blocked label candidates and which needs constant time to check if it contains a specific label candidate. So the runtime of this algorithm on average cases is just $O(n)$.

Due to its simplicity and good performance it does not support sophisticated optimization of an objective function. To a certain extent we can say that it pursues the objective of maximizing the number of labels because it tries to label as many point features as possible for the given candidates list order. Additionally the algorithm does not support any weighting. Label candidates with a higher priority are treated as any other label candidate. But as said, this algorithm was implemented for fast testing and as a reference

to the other algorithms that support optimization. Apart from that this algorithm produces acceptable solutions on instances with a low number of conflicts.

A feature that this algorithm supports is to keep an initial solution as much as possible. This is achieved by simply selecting the labels in the current solution first, before running through the rest of the full label list. The runtime of $O(n)$ is not altered by this feature.

6.5.5 `IndependentSet`-Algorithm

The *`IndependentSet`-Algorithm* we implemented in our framework searches for a maximum weight independent set. We use the *JGraphT*[9] library that provides graph structures and algorithms. Since there is no algorithm contained in the library to calculate a maximum weight independent set we use the `GreedyVCImpl` implementation to calculate a minimum weight vertex cover (or actually an approximation) that we then invert to get a maximum weight independent set or to get at least an approximation if the vertex cover is not of minimum weight.

This algorithm supports weights by nature but does not support the possibility to keep an initial solution. The runtime of this algorithm is determined by the slowest part of finding a minimal weighted vertex cover which is in $O(m \log n)$ for the used `GreedyVCImpl` algorithm, where n is the number of vertices and m the number of edges in the conflict graph.

6.5.6 `ThreeRules`-Algorithm

This algorithm is an implementation of the three rules approach proposed by Wagner et al. [WWKS01] and is described in Section 4.3 in more detail. Our implementation supports priorities of label candidates and pursues the objective of maximizing the total weight of the labels in the solution. If all the label candidates have the same weight, the number of labels is maximized (LNM). We also implemented a switch for this algorithm so that only phase one can be applied in order to reduce the size of the conflict graph. Since this algorithm modifies the conflict graph, the reduced version can then be used as input for the other algorithms too. The `ThreeRules` algorithm does not support the feature of keeping an initial solution.

Considering the time complexity of this algorithm, Wagner et al. [WWKS01] describe that (with some enhancements of rule three) it is in $O(n + k)$ where n is the number of candidates and k is the number of edges in the conflict graph. Since the number of candidates per feature is constant, the first two rules can be checked in constant time for each feature. A feature is put on the stack when one of its candidates was deleted or it has lost a conflict partner. Hence, this part of phase one sums up to $O(n+k)$. Rule three contains the check if the conflicting set of a candidate forms a clique which normally takes time quadratic in the number of conflict partners. But the authors describe that falling back on geometry it can be detected in linear time. With the additional enhancement to

[9] `http://jgrapht.org/` version 1.0.1

apply the rule only to candidates with less than a constant number of conflicts (since it is not very likely that the neighbourhood of a candidate with many conflicts forms a clique) it even can be checked in constant time. Phase two only needs linear extra time and hence the time complexity of this algorithm is said to be in $O(n + k)$.

Since it was not the aim of this thesis to create the best implementation of this algorithm we did not implement the enhancements of rule three for time reasons. Rule three is checked in our implementation in $O(k^2)$ time.

6.5.7 Comparison of the Algorithms

In this section we want to compare the implemented algorithms available in our prototype. Table 6.2 shows an overview of the algorithms, their objective function, their support for an exact solution, support for weights and support to keep an initially calculated solution as good as possible.

Considering weights (or priorities) all of our algorithms except `Simple` support weights and hence have as first objective the maximization of the total weights in the calculated solution (indicated by `wMax`). If the labels all have the same weight the second objective is pursued. This is either the label number maximization (LNM) or the minimization of the number of conflicts between the labels while labelling all features (MNC).

We only have one exact algorithm (the `MaxHS`-Algorithm) which is based on SAT solving. Except this and our greedy `Simple` algorithm, all other algorithms are heuristics. As can be seen in Table 6.2, most of our algorithms currently do not support to keep an initial solution. Only the `MaxHS` and the `Simple` algorithm support this feature. For the `ILP`, the `MNC` and the `IndependentSet` algorithms we can add this functionality by adapting the weights as it is implemented in the `MaxHS` algorithm. For the `ThreeRules` algorithm it is a bit more complicated to add.

Algorithm	objective function	exact	weights	keep solution
`ILP`	wMax + MNC	✗	✓	✗
`MaxHS`	wMax + LNM	✓	✓	✓
`MNC`	wMax + MNC	✗	✓	✗
`Simple`	LNM	✗	✗	✓
`IndependentSet`	wMax + LNM	✗	✓	✗
`ThreeRules`	wMax + LNM	✗	✓	✗

Table 6.2: A comparison of the implemented algorithms.

CHAPTER 7

Evaluation

In this chapter we present the experimental evaluation of our implemented prototype, the implemented labeling algorithms and their combinations with randomized label modifications. In the first section (Section 7.1) we describe the test setting and in the second section (Section 7.2) we present the actual results and discuss them.

7.1 Setting

Our test scenario can be described as follows. After parsing an input file we first apply one of our four label number maximization algorithms (`IndependentSet`, `MaxHS`, `Simple` and `ThreeRules`) to create an initial solution S_1. Then we randomly choose a number of labels from the solution (in our tests 20 percent of the solution size) and apply to this set M one of our modifications. We will use the font size shrink (i.e., label shrink) and the font size enlargement (i.e., label enlargement) since they are the most interesting modifications here. They represent the removing and adding of edges from and to the conflict graph. We shrink or enlarge the font size by four pixel. The initial and default font size of the labels is thirteen pixel. We further look at combinations of modifications meaning that we choose a subset of set M and apply to 0, 25, 50, 75 or 100 percent of labels in M the font size shrink and to the rest the font size enlarge. Then, applying another possibly different algorithm we get a new valid (i.e., a conflict free) solution S_2. We run this setting of calculating an initial solution, modifying it and recalculating a new one in a loop for a hundred times to avoid one-time artefacts and get better result data.

We investigate how the number of labeled features in S_2 differs dependent to the applied algorithm combinations and want to answer questions like: Do optimal algorithm combinations differ between dense and sparse data sets? Does it make sense to apply an exact algorithm twice or is it advisable to combine the exact algorithm with a heuristic approach? Is the non optimizing and greedy algorithm we implemented of any relevance? How does it perform in combination with our heuristics and our exact algorithm?

data set	number of features	conflicts	conflicts/node
Salzburg	137	3700	6,75
	292	15176	12,99
	473	33442	17,68
	601	50590	21,04
Lower Austria	350	6004	4,29
	531	10362	4,88
	863	15928	4,61
	1296	31108	6,00

Table 7.1: Our test data sets and their properties.

First of all we expect that the greedy algorithm `Simple` produces non competitive solutions when applied as first algorithm because of its missing optimizations. But it probably can produce sufficient solutions when applied as the second algorithm since it can benefit from the fact that it tries to keep a given solution as good as possible. Besides this it will be of course the fastest of all algorithms. We also expect that the exact algorithm results in the best solutions in terms of maximum number of labeled features but that it is very time consuming. Maybe a combination with a fast heuristic is best for both dense and sparse data sets.

Clearly this test scenario will not happen in real use of our application, since it recalculates a new solution after each modification. But this test scenario is of interest since it shows the effects of the modifications to the conflict graph respectively the solution recalculation better than just modifying one single label candidate. Additionally it shows which combinations of algorithms are well performing in the sense of solution quality as well as time used.

Like we did for our application we also use real world data from OSM for the evaluation. We therefore have prepared two different regions of Austria. One is the city region of Salzburg to get a quite dense data set and the second one is a rural region namely a part of Lower Austria to get a sparser region and a sparser conflict graph but with a higher number of features. We will call them *dense* and *sparse* data sets in the following. We also filtered and rehashed these data sets to get different graduations of density and feature amount with the same data. So in total we have eight different data sets where we applied our tests. Table 7.1 shows our test data sets with their density, their number of conflicts and their feature amount. As can be seen, the Lower Austria data sets have similar low numbers of conflicts per node and only increase in their number of point features. The zoom level for all of our tests is set to a zoom level of 16. This corresponds

to a map scale of 1:8000.

The tests are implemented like our prototype in Java and we ran them on a standard laptop system with 12GB RAM and an Intel®Core™i5-5200U @ 2.20 GHz, using a 64-Bit Windows 10 as the operating system.

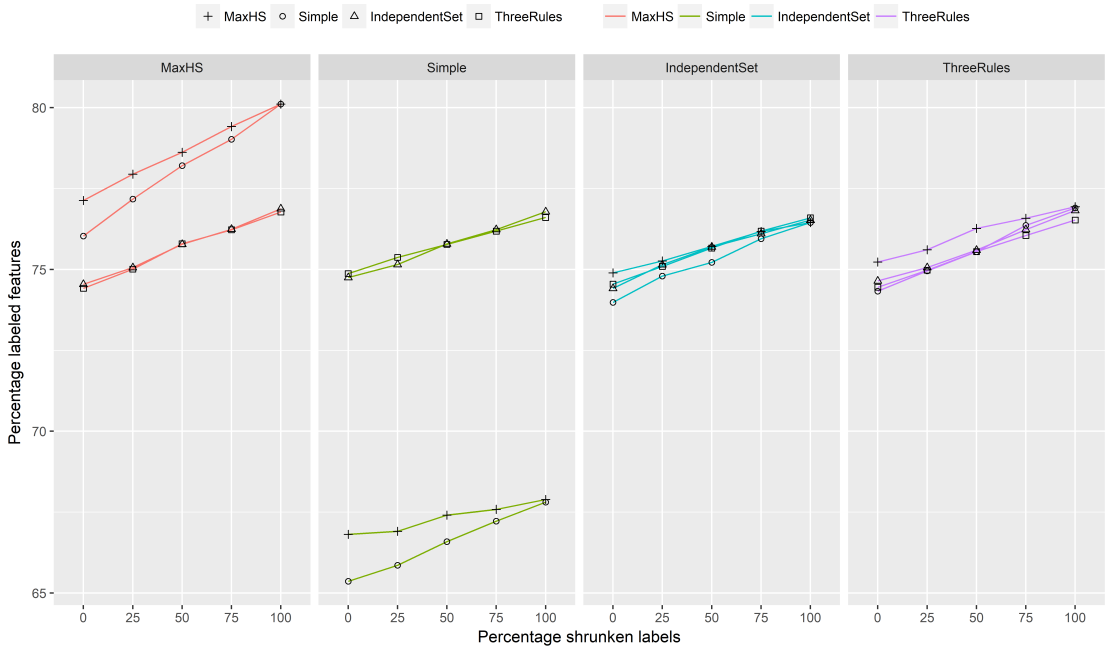
7.2 Results

The summary of our test runs is shown in Figure 7.1a and Figure 7.1b. They have to be interpreted as follows. Each colour (or each facet) represents the algorithm that was used for calculating the initial solution. The modification that was used is labeled on the x-axis where, e.g., a value of '25' indicates that 25 percent of the selected labels were reduced in their size while the remaining 75 percent were enlarged. The different point shapes indicate the second algorithm that was used to recalculate a new solution. The y-axis then represents the percentage of features that were labeled in the final solution by the specific algorithms and modification combinations. These values are the median values of all aggregated data sets of one density (i.e., all Salzburg data sets or all Lower Austria data sets) each containing one hundred runs. As can be seen in both figures, the percentage of labeled features increases when increasing the amount of labels that get shrunk while simultaneously decreasing the amount of enlarged labels. The rightmost value is the one where the size of all selected labels is reduced and hence its obvious that more features can be labelled.

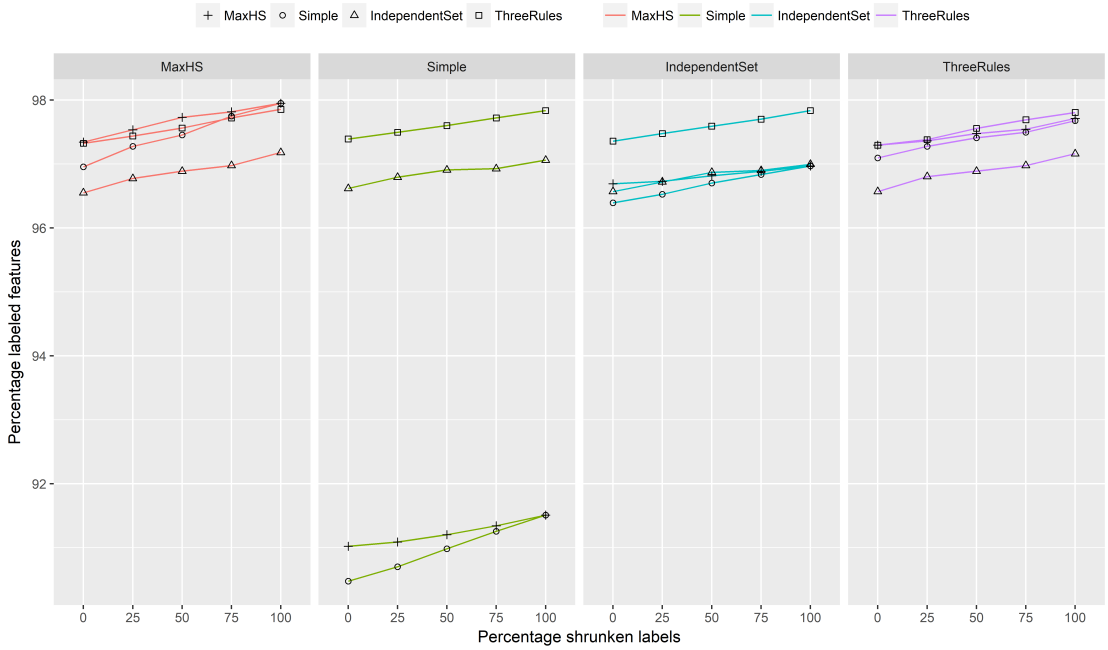
One striking but expected detail is that on both, the dense and the sparse data set, the `Simple` algorithm produces worse first solutions compared to the other algorithms. Even when we apply the exact algorithm as the second algorithm the solution stays suboptimal since this algorithm tries to keep the initial solution as good as possible. Different from that the `ThreeRules` and the `IndependentSet` algorithms ignore a solution calculated by a previous algorithm and hence result in quite competitive solutions when applied after the greedy `Simple` algorithm.

Starting with an exact solution calculated by `MaxHS` we can observe the opposite. Then, using `MaxHS` or `Simple` as second algorithm produces better solutions than the other two. But this effect just occurs on dense data sets. On our sparse data sets there are no significant differences except that the `IndependentSet` algorithm performs a little worse. An interesting fact is that applying the exact algorithm twice does not lead to much better solutions in the sense of more labels placed than applying the combination of first `MaxHS` and then `Simple` which is much faster than the first variant. Especially when all selected labels get shrunk this combination produces equally good results. This is because both want to keep the initial solution as good as possible. In general, using `MaxHS` for the recalculations on a sparse data set is not the best option, since it produces only in one case (i.e., applying twice the `MaxHS` algorithm) a slightly better solution then other faster algorithm combination.

Besides the above mentioned remarks there are no large differences between the other

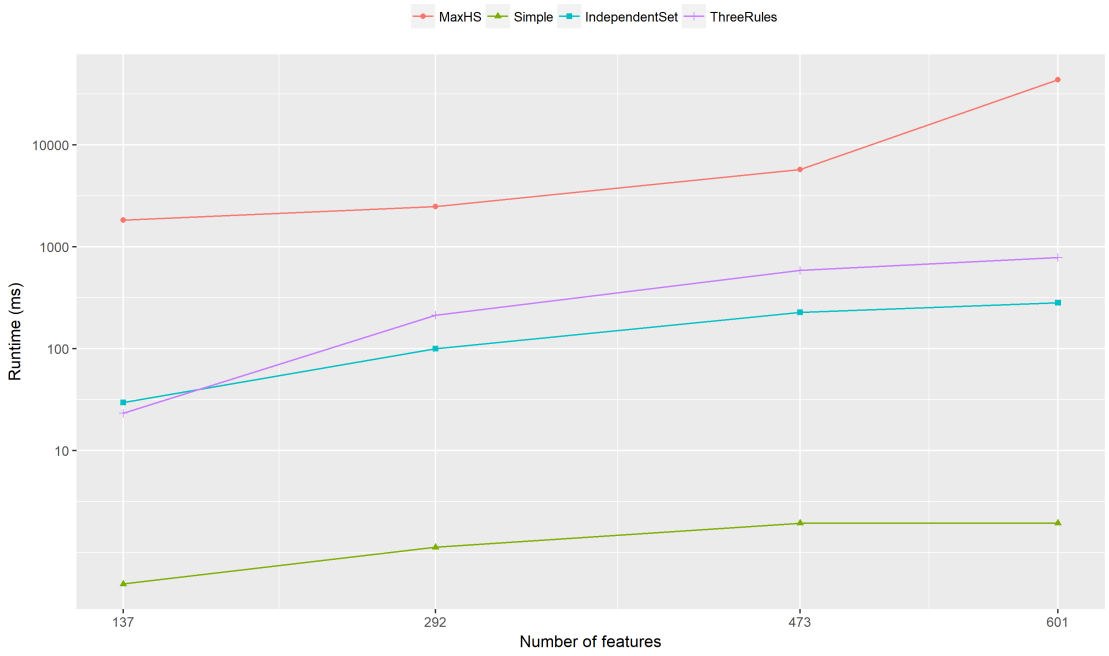


(a) Dense Salzburg data set.

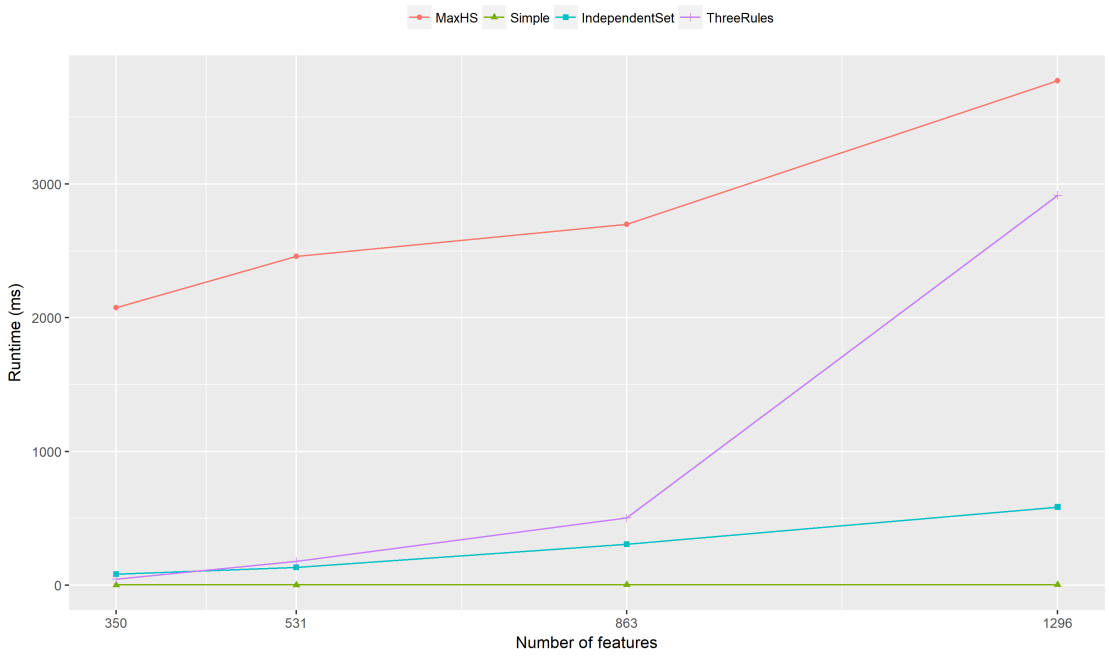


(b) Sparse Lower Austria data set.

Figure 7.1: Different algorithm and modification combinations. The x-axis shows the percentage of labels that were reduced in their size (the rest were enlarged) and the y-axis shows the percentage of features that could be labeled in the final solution.



(a)



(b)

Figure 7.2: Runtimes of the implemented algorithms for the (a) dense Salzburg and for the (b) sparse Lower Austria data sets. We use a logarithmic scale for the dense data set.

algorithm combinations on the dense data set (see Figure 7.1a). They all produce quite competitive labeling solutions. We can just point out the running time performance differences which can be seen in Figure 7.2a and Figure 7.2b for the two data sets. Note that we have chosen a logarithmic scale for the dense data set because of the long time needed for running the exact `MaxHS` algorithm. As expected the `Simple` algorithm is the fastest and needs just a few milliseconds even for large data sets. `IndependentSet` and `ThreeRules` are quite fast on small number of features whereas the latter is always a bit slower. But with growing feature number the `ThreeRules` algorithm needs more and more time and can not keep pace with the `IndependentSet` algorithm. The slowest one is for sure the exact algorithm on both dense and sparse data sets.

Looking at the sparse data set in Figure 7.1b we can observe that the `ThreeRules` algorithm outperforms the others when used for recalculation regardless which algorithm was used for calculating the initial solution. As a little side note we have to say that the `ThreeRules` modifies the conflict graph during its application so that we have to create the conflict graph from scratch before we can apply another algorithm. But fortunately this influences the overall runtime only minimally. An interesting fact for sparse data sets is that the `IndependentSet` algorithm performs worse than the others both when used as first and as second algorithm. Even the `Simple` algorithm outperforms it in some combinations.

Quite surprising is the combination of `Simple` with `ThreeRules`. It produces in most cases better solutions than any other combination using `ThreeRules` as the second algorithm (see Figure 7.3) although the differences are quite minimal.

So as we have seen we can not identify the one combination for all kind of data sets. But we can identify well performing combinations for specific use cases and specific conditions. Table 7.2 shows some use cases and the best algorithm combination for it. For example if the user always wants the best solution and is not interested in fast results but has time in his workflow to wait for the solutions, he will obviously use the exact algorithm twice. Much faster but also a bit less optimal variant for dense data sets is the combination of the exact algorithm with the greedy algorithm `Simple` as second algorithm. If time is the priority constraint for the user he can combine the `Simple` algorithm with either the `ThreeRules` algorithm or if he works on large data sets he can combine it with the `IndependentSet` algorithm because as we already said the `ThreeRules` algorithm gets much slower on large data sets.

As we can see in Table 7.2 it is a quite good choice to use the greedy and simple algorithm either as first or second algorithm but not twice. This is an interesting point that we did not expect, since it is an algorithm without any optimization. But the fact that it can keep an existing solution and the speed of its calculations makes it an interesting algorithm for several use cases.

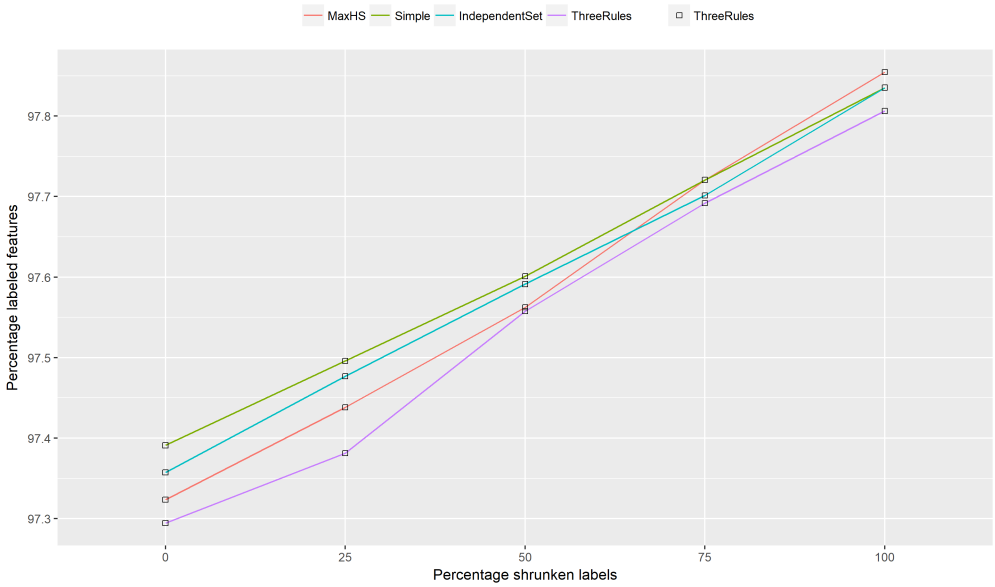


Figure 7.3: Comparison of the algorithms when using `ThreeRules` as second algorithm on the sparse data set.

Constraints	Data set	Algorithm combination
optimal, no time constraints	dense or sparse	`MaxHS+MaxHS`
optimal, fast	dense	`MaxHS+Simple`
	sparse	`Simple+ThreeRules` or `IndependentSet+ThreeRules`
good solution, fastest	dense or sparse	`Simple+ThreeRules` or `Simple+IndependentSet`

Table 7.2: Different use case constraints and suitable algorithm combinations (the ordering indicates which one is used as first and which one as second algorithm).

Fine Grained Evaluation

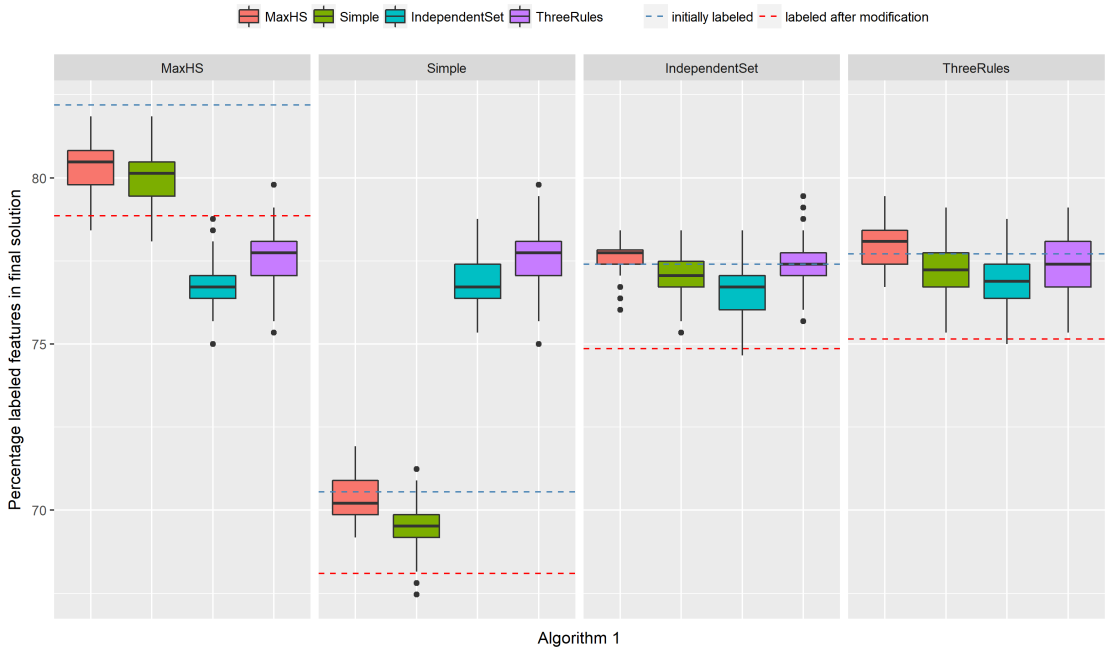
As we have said, in the above plots all data sets of one density per modification are aggregated. Since there is some information lost we show here some selected (representative) plots of different data sets and different modifications (see Figures 7.4 and 7.5). These boxplots have to be read as follows. Each facet represents the algorithm that was used to calculate the initial solution. The different coloured boxes then indicate the algorithm that was used for the recalculation and on the y-axis we again have the percentage of labeled features in the final solution by the specific algorithm combination. Each plot

represents one modification, e.g., Figure 7.4b is the plot for shrinking and enlarging each 50 percent of the selected labels in the dense Salzburg data set with 601 features.

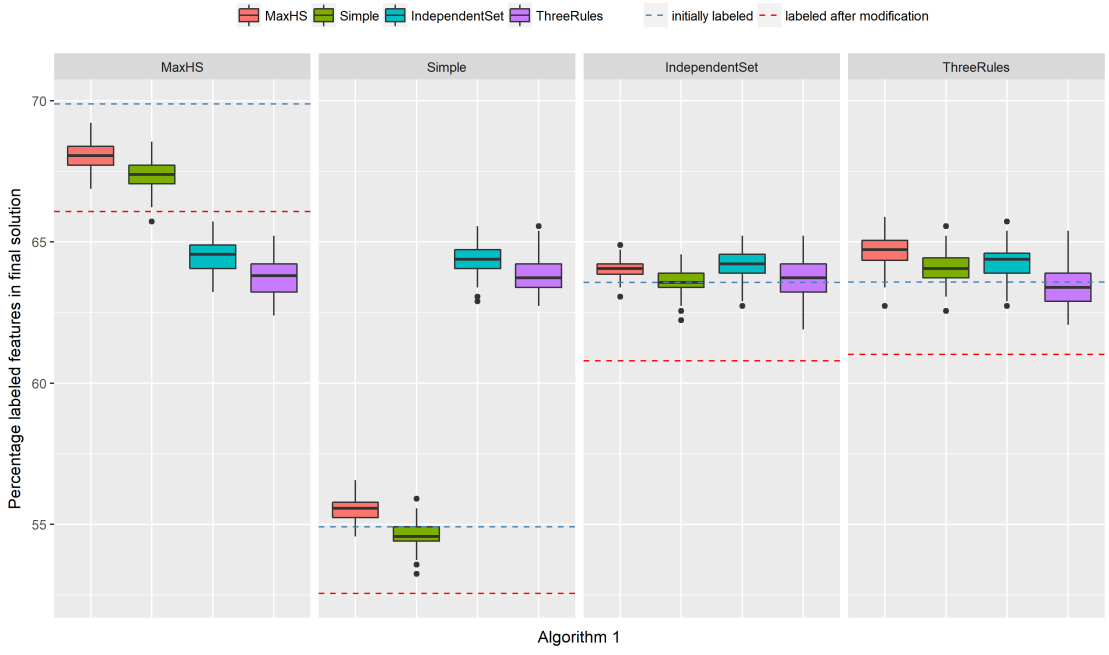
First looking on the size of the initial solution (the blue dashed line in the figures) we can see that for sure the exact `MaxHS` algorithm produces the best initial solutions and the simple and greedy algorithm produces initial solutions that have more than ten percent (in the dense data set) and six to eight percent (in the sparse data set) less features labeled than the exact algorithm. Comparing the `IndependentSet` and the `ThreeRules` algorithms we can see that those algorithms perform equally good on dense data sets and calculate initial solutions that have about five to seven percent less features labeled than the exact algorithm. On sparse data sets the `ThreeRules` algorithm outperforms the `IndependentSet` algorithm by one percent and produces solutions near to optimal especially on sparse data sets with lower numbers of features, i.e., in our tests for all data sets with 863 features or less (see Figure 7.5a).

The red dashed line in the plots indicates the solution that is created by simply removing all conflicting labels after the modification was performed. It is noticeable that some algorithm combinations even calculate solutions that have less features labelled than this simple conflict solving/removing. Drastic examples are when the `IndependentSet` or the `ThreeRules` algorithms are applied to a solution calculated by `MaxHS` (see Figures 7.4a and 7.4b). This is because these two algorithms ignore a previously calculated solution.

Figures 7.5a and 7.5b show clearly the dominance of the `ThreeRules` algorithm when used as the second algorithm on sparse data sets. And on dense data sets it even depends on the number of features which one of the `IndependentSet` and the `ThreeRules` algorithm is better. In Figure 7.4a the `ThreeRules` algorithm is ahead of the `IndependentSet` and in Figure 7.4b it is inversely. Another aspect we can see is that these two algorithms ignore the initial solution of the first algorithm and hence produce similar good solutions regardless which algorithm was performed before. This is also the reason why their solutions are much better than an initial solution calculated by `Simple`. In this case the `MaxHS` algorithm is tied to that solution and can not keep pace with them in this combination. For the update idea of our framework where we want to keep the initial solution as much as possible, the `IndependentSet` and the `ThreeRules` algorithms are not suitable when used as second algorithm since they ignore a previously calculated solution. This can lead to solutions that are very different from the initial solution. `MaxHS` combined with `Simple` does not have this drawback.

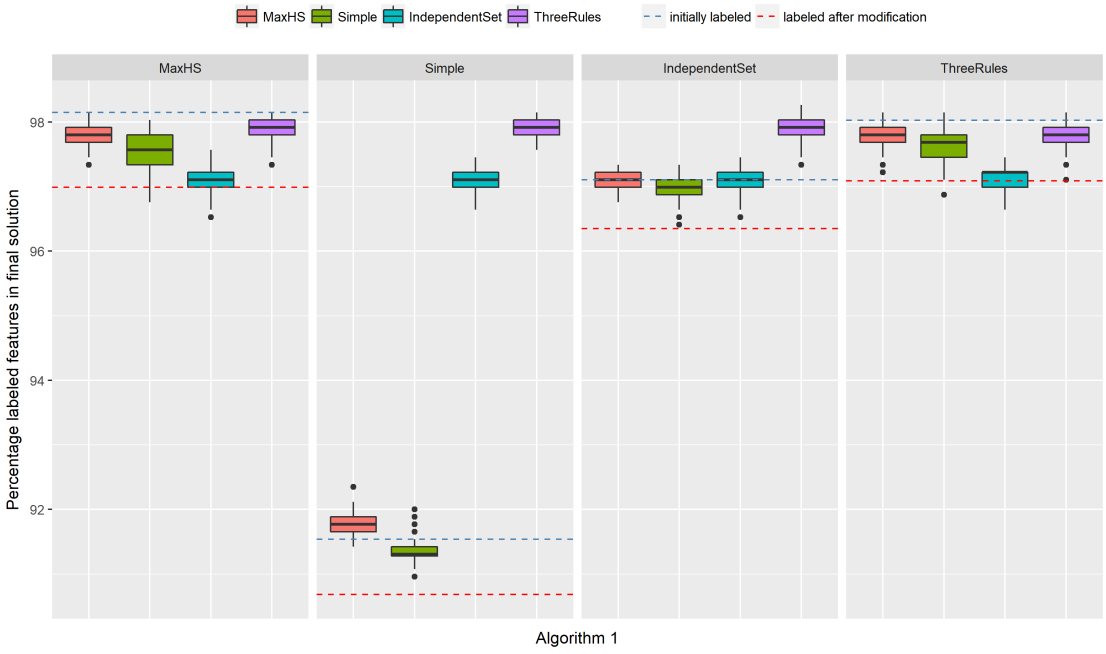


(a)

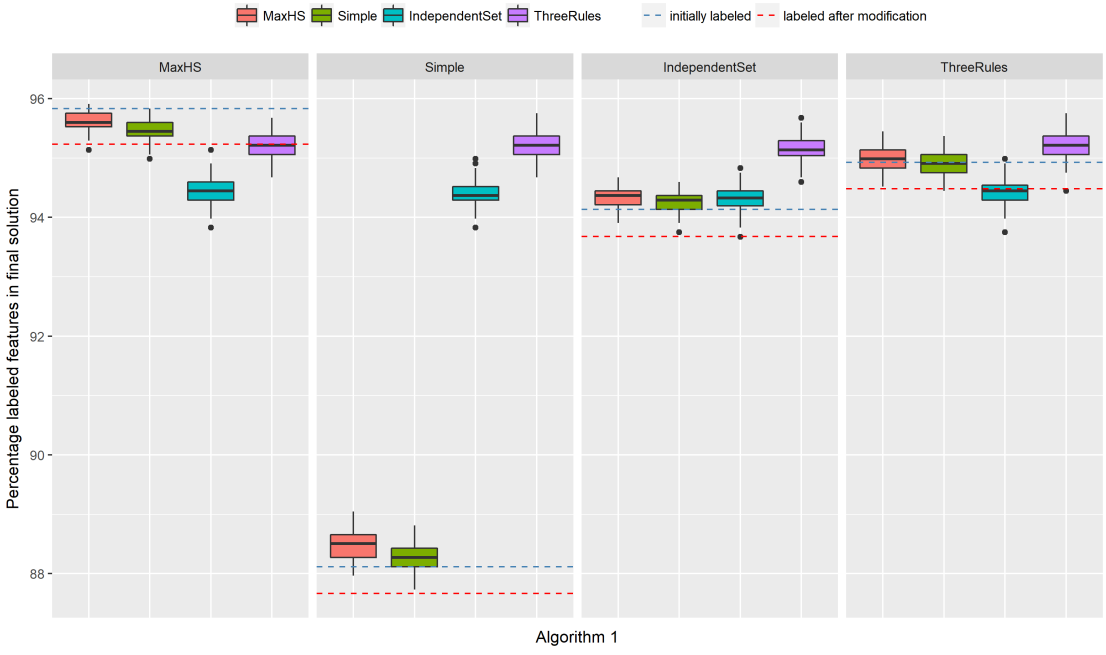


(b)

Figure 7.4: Algorithm combinations for shrinking and enlarging each 50 percent of the selected labels in the dense Salzburg data set with (a) 292 and (b) 601 features.



(a)



(b)

Figure 7.5: Algorithm combinations for shrinking (a) 25 and (b) 75 percent of the selected labels in the sparse Lower Autria data set with (a) 863 and (b) 1296 features.

CHAPTER 8

Conclusion

In this thesis we studied a semi-automatic and user-centered approach for labeling point features on static cartographic maps. We presented an interactive labeling tool based on user constraints. Beginning with an initial solution the domain expert user can add his or her domain knowledge in the form of user constraints to the tool and so refine, adapt and improve the labeling solution to create a high quality map.

We incorporated domain expertise collected in interviews with a domain expert to create a software regarding the users needs. We use a conflict graph to formally describe the user constraints and the labeling problem in general and we use it for our algorithms to calculate valid and optimal labeling solutions. The advantage of this approach is that we are not tied to a specific type of feature, a position model or a specific kind or shape of labels although we currently use the fixed four position model for rectangular axis parallel label candidates. We offer three different optimization approaches regarding automatic labeling, namely weight maximization, label maximization and conflicts minimization. In this thesis we focused on the label number maximization.

In an evaluation of our labeling prototype and our algorithms with real world data from OpenStreetMap we showed that different algorithm combinations perform best based on the input data set, the modification that is applied as well as the goals and constraints a user is interested in. We could show differences between dense and sparse data sets and gave suggestions for specific requirements of the creation process (e.g., optimal solution, fast results). We have seen that our heuristics perform similarly well or even better than the exact algorithm in the use cases of our application. For sure the exact algorithm generates the best initial solution but in connection with the label modification and the recalculations it does not always outperform the implemented heuristics. Additionally it needs much more time to produce a solution. So the experiments showed, that it is not necessary or advisable to always apply an exact algorithm to get the best solutions, but that it is sufficient and not less optimal to use the faster heuristics. We have also seen,

that we can even use the very fast greedy algorithm with no optimization for calculating or recalculating a solution and get quite competitive results.

One of our intentions for this thesis was to find techniques where we do not need to recalculate the whole labeling after the user added a constraint by modifying the initial solution. We tried to keep the changes in the conflict graph local and wanted to update just the neighbourhood of where the modification happened to achieve an efficient recalculation. For that we searched for some patterns and approaches that allow us to keep the effects of a modification in some restricted area but unfortunately we did not succeed here. We left out this aspect for now and for this thesis but we will pursue this goal in future work. Actually, this problem does not need to be tied to the labeling problem but can be seen as a more general problem in any undirected graph in connection with some "solution" (i.e., a subset of graph vertices) like a (maximum) independent set or a (minimum) vertex cover. The question that arises is: how can we keep a graph modification (i.e., adding or removing an edge) local and influencing as little as possible the given set of vertices?

We see this thesis as a first step and want to extend it further by, e.g., supporting line and area features or improve the process of modification and recalculation including the above mentioned local updates. If these changes are fast we can also extend the framework by giving previews to the user of the effects a particular modification will have or warn of modifications with wide-spread consequences. Another possibly interesting approach is to compute small sets of distinct solutions with comparable quality values, where the user can choose one.

We can also expand our framework by adding more algorithms or improve the existing ones. For example we can try to optimize the `Simple` algorithm or adapt the `IndependentSet` and the `ThreeRules` algorithms so that they also take an initial solution into account. Our current evaluation does not say anything about the percentage of labels that are kept by a specific algorithm when calculating a new solution after a label modification was applied. At the moment such an evaluation would be a bit distorted because of the above mentioned missing support in the `IndependentSet` and `ThreeRules` algorithms. But as we have tried to aim for algorithms that keep a previously calculated solution as much as possible this is an interesting aspect that can be evaluated in future work.

Bibliography

[ABL10] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial maxsat. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pages 3–8. AAAI Press, 2010.

[AI16] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016.

[AKCF+04] Faisal N. Abu Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In Lars Arge, Giuseppe F. Italiano, and Robert Sedgewick, editors, *ALENEX/ANALC*, pages 62–69. SIAM, 2004.

[ArDT09] Adriana C.F. Alvim and Éric D. Taillard. POPMUSIC for the point feature label placement problem. *European Journal of Operational Research*, 192(2):396 – 413, 2009.

[AS95] Walid G. Aref and Hanan Samet. A window retrieval algorithm for spatial databases using quadtrees. In *Proceedings of the 3rd ACM International Workshop on Advances in Geographic Information Systems, Baltimore, Maryland, December 1-2, 1995, in conjunction with CIKM 1995.*, page 69, 1995.

[AvKS98] Pankaj K. Agarwal, Marc van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Computational Geometry*, 11(3):209–218, 1998.

[Bac] Tomislav Bacinger. Best online mapping tools for web developers. `https://www.toptal.com/web/the-roadmap-to-roadmaps-a-survey-of-the-best-online-mapping-tools`. Accessed: 2018-05-14.

[BBK98] Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegal. The pyramid-technique: Towards breaking the curse of dimensionality. *SIGMOD Rec.*, 27(2):142–153, June 1998.

[BDY06] K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):773–780, Sept 2006.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, May 1990.

[BKSW07] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry*, 36(3):215 – 236, 2007.

[BNPW10] Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry*, 43(3):312–328, 2010.

[BT07] Sergiy Butenko and Svyatoslav Trukhanov. Using critical sets to solve the maximum independent set problem. *Operations Research Letters*, 35(4):519–524, 2007.

[CMS95] Jon Christensen, Joe Marks, and Stuart Shieber. An empirical study of algorithms for point-feature label placement. *ACM Trans. Graph.*, 14(3):203–232, July 1995.

[CRL08] Gildásio Lecchi Cravo, Glaydston Mattos Ribeiro, and Luiz Antonio Nogueira Lorena. A greedy randomized adaptive search procedure for the point-feature cartographic label placement. *Comput. Geosci.*, 34(4):373–386, April 2008.

[DB11] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Lee, editor, *Principles and Practice of Constraint Programming – CP 2011*, LNCS 6876, pages 225–239. Springer Berlin Heidelberg, 2011.

[dBvKOS00] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Quadtrees*, chapter 14, pages 307–322. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[dNE08] Hugo A.D. do Nascimento and Peter Eades. User hints for map labeling. *Journal of Visual Languages & Computing*, 19(1):39–74, 2008.

[EGS05] David Eppstein, Michael T. Goodrich, and Jonathan Z. Sun. The skip quadtree: A simple dynamic data structure for multidimensional data. In *Proceedings of the Twenty-first Annual Symposium on Computational Geometry*, SCG '05, pages 296–305, New York, NY, USA, 2005. ACM.

[FB74] R. A. Finkel and J. L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, Mar 1974.

[FR89] Thomas A Feo and Mauricio G.C Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67 – 71, 1989.

[FW91] Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *In Proc. 7th Annual ACM Symposium on Computational Geometry*, pages 281–288, 1991.

[Gar82] Irene Gargantini. An effective way to represent quadtrees. *Commun. ACM*, 25(12):905–910, December 1982.

[GD06] Stephen Gilmour and Mark Dras. Kernelization as heuristic structure for the vertex cover problem. In Marco Dorigo, Luca Maria Gambardella, Mauro Birattari, Alcherio Martinoli, Riccardo Poli, and Thomas Stützle, editors, *Ant Colony Optimization and Swarm Intelligence*, LNCS 4150, pages 452–459. Springer Berlin Heidelberg, 2006.

[geo] Geofabrik Download Server. `http://download.geofabrik.de/`. Accessed: 2018-05-16.

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

[GNR11] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Consistent labeling of rotating maps. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, LNCS 6844, pages 451–462, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[HP05] Sariel Har-Peled. Quadtrees-hierarchical grids. *Lecture notes*, 2005. `http://sarielhp.org/teach/2004/a_aprx/lec/03_quadtree.pdf` Accessed: 2018-08-06.

[Imh75] Eduard Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.

[JC04] Joo-Won Jung and Kyung-Yong Chwa. Labeling points with given rectangles. *Inf. Process. Lett.*, 89(3):115–121, February 2004.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311. ACM, 1984.

[Kla01] Gunnar W. Klau. *A Combinatorial Approach to Orthogonal Placement Problems*. PhD thesis, Saarland University, 2001.

[KP07] You Jung Kim and Jignesh M. Patel. Rethinking choices for multi-dimensional point indexing: Making the case for the often ignored quadtree. In *CIDR*, pages 281–291, 2007.

[KT98] Konstantinos G. Kakoulis and Ioannis G. Tollis. A unified approach to labeling graphical features. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 347–356, New York, NY, USA, 1998. ACM.

[LNS16] Maarten Löffler, Martin Nöllenburg, and Frank Staals. Mixed map labeling. *J. Spatial Information Science*, 13:3–32, 2016.

[map] The Map-Labeling Bibliography. `http://i11www.iti.kit.edu/~awolff/map-labeling/bibliography/`. Accessed: 2018-04-11.

[max] MaxSAT Evaluation 2017. `http://mse17.cs.helsinki.fi/mse17-talk.pdf`. Accessed: 2018-04-30.

[Mea82] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[MRL10] Geraldo R. Mauri, Glaydston M. Ribeiro, and Luiz A.N. Lorena. A new mathematical model and a lagrangean decomposition for the point-feature cartographic label placement problem. *Computers & Operations Research*, 37(12):2164 – 2172, 2010.

[MS91] Joe Marks and Stuart Shieber. The computational complexity of cartographic label placement. Technical report, Harvard Computer Science Group, 1991.

[NHS84] J. Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, March 1984.

[NN16] Benjamin Niedermann and Martin Nöllenburg. An algorithmic framework for labeling road maps. In Jennifer A. Miller, David O'Sullivan, and Nancy Wiegand, editors, *Geographic Information Science*, pages 308–322, Cham, 2016. Springer International Publishing.

[odb] Open Database License (ODbL) v1.0 | Open Data Commons. https://opendatacommons.org/licenses/odbl/1.0/. Accessed: 2018-05-15.

[osm] OpenStreetMap. www.openstreetmap.org. Accessed: 2018-05-15.

[pla] Planet OSM. https://planet.openstreetmap.org/. Accessed: 2018-05-16.

[PM98] F Pascal and JL Marechal. Fast adaptive quadtree mesh generation. In *7th International Meshing Roundtable*, pages 211–224. Citeseer, 1998.

[RL06] Glaydston Mattos Ribeiro and Luiz Antonio Nogueira Lorena. Heuristics for cartographic label placement problems. *Computers & Geosciences*, 32(6):739–748, 2006.

[RL08] Glaydston Mattos Ribeiro and Luiz Antonio Nogueira Lorena. Lagrangean relaxation with clusters for point-feature cartographic label placement problems. *Computers & Operations Research*, 35(7):2129 – 2140, 2008.

[RML11] Glaydston M. Ribeiro, Geraldo R. Mauri, and Luiz Antonio N. Lorena. A Lagrangean decomposition for the maximum independent set problem applied to map labeling. *Operational Research*, 11(3):229–243, 2011.

[RR10] Mauricio G.C. Resende and Celso C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 283–319, Boston, MA, 2010. Springer US.

[RR17] Maxim Rylov and Andreas Reimer. A practical algorithm for the external annotation of area features. *The Cartographic Journal*, 54(1):61–76, 2017.

[Sam84] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, June 1984.

[Sam90] Hanan Samet. Applications of spatial data structures. In *Computer Graphics, Image Processing, and GIS*, pages 0–201. Addison-Wesley, 1990.

[sat] SAT 2017. http://sat2017.gitlab.io/. Accessed: 2018-04-30.

[SB94] G. J. Sullivan and R. L. Baker. Efficient quadtree coding of images and video. *IEEE Transactions on Image Processing*, 3(3):327–331, May 1994.

[SRSW84] Hanan Samet, Azriel Rosenfeld, Clifford A. Shaffer, and Robert E. Webber. A geographic information system using quadtrees. *Pattern Recognition*, 17(6):647 – 656, 1984.

[Str16] Darren Strash. On the power of simple reductions for the maximum independent set problem. In Thang N. Dinh and My T. Thai, editors, *Computing and Combinatorics*, LNCS 9797, pages 345–356. Springer International Publishing, 2016.

[SVA00] Tycho Strijk, Bram Verweij, and Karen Aardal. Algorithms for maximum independent set applied to map labelling. Technical report, Utrecht University, 2000.

[swe] OpenStreetMap Sverige. `http://openstreetmap.se`. Accessed: 2018-05-14.

[VA99] Bram Verweij and Karen Aardal. An optimisation algorithm for maximum independent set with applications in map labelling. In Jaroslav Nešetřil, editor, *Algorithms - ESA' 99*, LNCS 1643, pages 426–437. Springer Berlin Heidelberg, 1999.

[vKSW98] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point set labeling with sliding labels. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 337–346. ACM, 1998.

[wgs] WGS 84 / Pseudo-Mercator. `https://epsg.io/3857`. Accessed: 2018-05-16.

[WKvK+00] Alexander Wolff, Lars Knipping, Marc van Kreveld, Tycho Strijk, and Pankaj K. Agarwal. A simple and efficient algorithm for high-quality line labeling. In Peter M. Atkinson and David J. Martin, editors, *Innovations in GIS VII: GeoComputation*, chapter 11, pages 147–159. Taylor & Francis, 2000.

[WWKS01] F. Wagner, A. Wolff, V. Kapoor, and T. Strijk. Three rules suffice for good label placement. *Algorithmica*, 30(2):334–349, June 2001.

[Yoe72] Pinhas Yoeli. The logic of automated map lettering. *The Cartographic Journal*, 9(2):99–108, 1972.

[Zor86] Steven Zoraster. Integer programming applied to the map label placement problem. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 23(3):16–27, 1986.

[Zor90] Steven Zoraster. The solution of large 0–1 integer programming problems encountered in automated cartography. *Operations Research*, 38(5):752–759, 1990.