Chapter 1

An Introduction to Label Placement

In everyday life, we permanently categorize and label things or people according to the categories into which they in our opinion fall. If we do not know somebody's name, we may refer to him by his physical appearance, his hair style, profession or possessions, the way he dresses, behaves, or talks. That is, we label him as "tall", "brunette", "poor", "extroverted", "southern", or with several of these predicates. If we accumulate enough labels, we get a unique description. We use labels to identify, describe or simply store something in our memory. Labels never catch all information available on an object but rather focus on features that distinguish it from others. Spoken in terms of computer science, a label can be seen as a hash key that allows us to access additional information about the labeled object in the dictionary, which is represented by our brain.

We use labels to describe objects and to communicate our ideas to others, hoping that the hash key works in their dictionaries as well as in ours. In order to illustrate our ideas, we often resort to images, i.e. two-dimensional mappings of reality. These tend to consist of rather simple, and thus abstract graphical elements, which can have an abundance of meanings. Thus we must annotate these objects with some kind of labels to clarify our intentions. Such a labeling must fulfill two important requirements, namely (a) legibility, i.e. a label must be of sufficient size and must neither overlap objects of the image nor other labels, and (b) unambiguity, i.e. it must be clear which object a label annotates. The second requirement is also valid for the use of labels in speech, while the first results from the geometric limitations of the plane. From now on we will refer to objects that are to be labeled as *features*.

Labeling is one of the key tasks in the process of information visualization. In diagrams, maps, technical or graph drawings, features like points, lines, and polygons must be labeled to convey information. The interest in algorithms that automate this task has increased with the advance in type-setting technology and the amount of information to be visualized. Cartographers, graph

drawers, and computational geometers have suggested solutions to the labeling problem such as expert systems, 0-1 integer programming, approximation algorithms, and simulated annealing to name only a few. The ACM Computational Geometry Impact Task Force report [C+96] denotes label placement as an important research area. To fully comprehend the interest in automated labeling systems, one has to realize that manually labeling a map, for instance, is estimated to take fifty percent of total map production time [Mor80].

1.1 Historic Development

Going back in history, cartography is probably the oldest field that combined graphical with textual elements and was thus forced to deal with the labeling problem. The first record of a scaled map was found in China and is estimated to be about 2,300 years old [SZ97].

In cartography, the basis for automation was laid in the early sixties when the prominent Swiss cartographer Eduard Imhof published a catalogue of rules for label placement including good and bad examples [Imh62, Imh75]. In the same year, but independently of Imhof, Georges Alinhac, "Artiste Cartographe Principale" at the French Institut Géographique National, published the book "Cartographie Théorique et Technique", which includes a similar set of labeling guidelines [Ali62]. These first formalizations of label placement have certainly facilitated the step from craftsmanship to technology. Before, based on his taste and long work experience, a cartographer could judge a map as being "well" or "poorly" labeled, but since the publication of Imhof's and Alinhac's rules, the deficiencies of a map labeling can be named in detail. Only then, after a model was found and the objectives were made clear, could technicians and researchers without expertise in cartography start to automate the process of label placement. This was crucial for the field since the apparently most basic problem of label placement, i.e. labeling points, turned out to be hard in terms of computational complexity.

Ten years after Imhof and Alinhac declared the principles of label placement, the Israeli cartographer Pinhas Yoeli wrote the first article dealing with the automation of map labeling [Yoe72]. He suggested an interactive system consisting of a human map editor, a geographical database, an output and a "principle-of-placement" module as well as a placement and an operational program. He devoted a lot of attention to the arrangement and the interplay between the parts of his system, which must be due to the limitations in storage, computation speed and the type-setting abilities of output devices at that time. He did not give any details of his placement program, but suggested that after each run of the program, the map editor would evaluate the results and decide whether his "preliminary estimate as to the name carrying abilities of his map was too optimistic". In that case "there will be names for which the computer could not find any place". If the map editor cannot add these manually, he has to revise decisions (concerning font size, place selection) and rerun the program until a satisfactory solution is found.

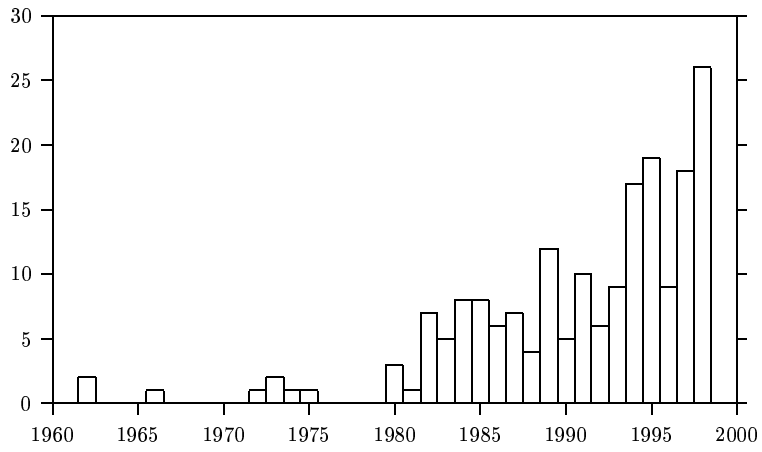

Figure 1.1: Number of publications over time in the label-placement literature.

Two articles by Boyle and a master's thesis by Wilkie at the Department of Electrical Engineering, University of Saskatchewan followed at the beginning of the seventies [Boy73, Wil73, Boy74], but it was not until the eighties that a larger number of researchers became interested in labeling problems, see Figure 1.1. The figure shows for each year the number of references contained by the *Map-Labeling Bibliography* [WS96], an exhaustive list of literature about (mostly automated) label placement. The Map-Labeling Bibliography was integrated into the *Collection of Computer Science Bibliographies* [Ach95] in 1998.

Since the beginning of the eighties, there has been a steady flow of publications about the labeling problem in fields as diverse as artificial intelligence, cartography, geography, geology, spatial data handling, database systems, data structures, image processing, graph drawing, and computational geometry. Over the years, there were two diverging lines of research. Given the complexity of the labeling problem, most publications were directed towards developing and improving solutions for special cases. Another approach was targeting at finding a general labeling framework.

1.2 Theory...

Theoreticians, mostly computational geometers, entered the field around 1990. Nearly all work in terms of approximation algorithms and NP-hardness results has been focussed on the point labeling problem. This may be due to the fact that there are obvious models and objective functions for this problem. Nevertheless, there has been a rather controversial debate among cartographers over the "right" point-labeling model [WB91, Mil94]. Usually labels are restricted to axis-parallel rectangles, for example the bounding boxes of place names, but squares in arbitrary positions and disks have also been considered [DMM+97]. These labels must (a) not intersect any other label, and (b) touch the point they

label. While condition (a) guarantees legibility (given labels of sufficient size), condition (b) is responsible for the unambiguity of a labeling. Condition (b) has often been further restricted to the case where one of a label's corners must coincide with the point to be labeled. Some cartographers additionally allow the four positions where the midpoint of a label edge coincides with the point. This is how the basic labeling requirements mentioned above are modeled for points.

However, even apparently simple special cases of the point labeling problem, like deciding whether a set of points can be labeled with unit squares in one of four positions, have turned out to be NP-hard [MS91, FW91, KR92]. Therefore most theoreticians have studied approximation algorithms that maximize either the label size [FW91, DMM$^+$97], the number of points with labels [AvKS98, vKSW99], or both criteria simultaneously [DMM$^+$97]. Nevertheless, algorithms that can solve problems of a few dozen to a few hundred points optimally have also been studied in the past [Pre93, KMPS93, Sch95, VA99].

1.3 ... and Practice

Practitioners on the other hand have proposed an abundance of models and heuristics for labeling point, linear or area features on maps, and nodes or edges in graph drawings. While most of these algorithms may work well in practice, they lack guarantees on the quality of their results and often even asymptotic bounds on their runtime.

If quality guarantees or time bounds cannot be given, it is important to compare algorithms experimentally, both on real-world and synthetic data. There are extensive experimental comparisons of point labeling algorithms [CMS95, CFMS97, WW98]. However, we are not aware of similar work on algorithms for labeling more complex features. The reason for this may be that models for labeling one- or two-dimensional features tend to include aesthetic criteria that are highly application dependent.

1.4 Quality

Only recently another, more fundamental, question has been treated; namely how the quality of a label placement can be defined [vDvKSW99]. The basic idea is to collect a set of rules similar to those proposed by Imhof, and to quantify these rules subsequently. Such a quantification would have the task to produce formulae with a small number of parameters that can be set according to the application. The formulae should in turn be designed so that they are easily computable. Given an automatic label-quality checker, it will be possible to evaluate existing label-placement algorithms, locate their deficiencies, and ultimately come up with better algorithms. Determining a set of evaluation functions, upon which the label-placement community would generally agree, would mean a significant progress of the field, a step from a technical to a

scientific level.

So far the most common criteria taken into account for judging the quality of a label placement were either the number of features labeled or the label size. Again, only for labeling points more elaborate quality criteria have already been proposed and implemented, usually with the help of genetic algorithms [Djo94, VWS97, Pre98, Rum98, Rai98, Rai99, vDTdB99]. An exception to this is a rule-based system that Anthony Cook proposed and implemented in Prolog in collaboration with the British Ordnance Survey [Coo88]. He explicitely lists Imhof's rules and takes many of them into consideration.

1.5 Future Development

Since Yoeli opened the field of literature on automated label placement, the speed of hardware and the quality of printers has increased dramatically while the price for storage has dropped by the same order of magnitude. Furthermore, data structures and algorithms even for complex geometric problems have become widely available through libraries like LEDA [NM90], CGAL [Ove96], or the STL [MS96].

Concerning the future of label placement, I think that we can expect development into two somewhat contrary directions, namely high-quality and on-line labeling.

Assuming that the performance of computers and the price of human labour will further increase, so will the interest in high-quality labeling systems. So far, the only commercially available product is MAPLEX. This system was initially developed by a team of researchers under Christopher Jones at the University of Glamorgan, Wales [JC89]. Later the development of MAPLEX was continued by a company that was recently bought by one of the large producers of geographic information systems.

The core functionality of geographic information systems, or GIS for short, is to allow geographic data to be easily linked with other layers of information. Since GIS became available on PCs, they have gained enormous popularity for all kinds of administrative and planning tasks. One would think that labeling is a prominent feature of such systems, but so far most GIS offer only very basic placement routines. In practice, a GIS user is still forced to invest several hours in order to eliminate manually all label-label and label-feature intersections on a map—in spite of the large number of publications on automated label placement.

Given the success and the increasing availability of the Internet, the other important string of development can be expected to focus on rather simple, but very fast on-line applications. On-line mapping and label placement will certainly benefit from future extensions of the *hypertext mark-up language* (HTML). Soon it will be possible to transmit and depict vector images including text instead of bitmaps that tend to consume a lot of transmission time and computer storage. Already the current browser generation is able to place

text on top of graphics. This is one of the features of a much broader concept, namely *cascading style sheets* [JT98]. However, so far there is no generally accepted standard for font metrics. Thus the length of a textual label can vary from browser to browser, which makes it impossible to avoid intersections.

1.6 Overview

With this thesis, I would like to help narrowing the gap between theory and practice in automated label placement by presenting research in both directions. The thesis deals with the general label-placement problem, then investigates how to label points with rectangles or circles, how to label polygonal lines like rivers and finally how to design flexible geometric algorithms.

1.6.1 General Labeling, Compatible Representatives, and CSP

The general label-placement problem consists of labeling a set of *features* (points, lines, regions) given a set of *label candidates* (rectangles, circles, ellipses, irregularly shaped labels) for each feature. Each feature and each of its label candidates has a specified position in the plane. In general, a *label placement* or *labeling* simply specifies a subset of the features and choses for each of these features a *label* from its set of label candidates such that no two labels intersect. In a *complete labeling* all features receive labels. Deciding whether a complete labeling exists is NP-hard in general [MS91, FW91]. Therefore, researchers have turned their attention mainly to develloping heuristics and approximation algorithms for two obvious optimization versions of the problem, namely *label-number maximization* and *label-size maximization*.

The decision problem is a special case of the *problem of compatible representatives* introduced by Knuth and Raghunathan [KR92]. Label candidates are compatible representatives of their features if they do not intersect. In other words, we restrict compatibility to the geometric meaning implied by our context. Knuth and Raghunathan point out that "cartographers face an interesting case of the problem of compatible representatives". The authors suggest that "it seems worthwhile to add the problem of compatible representatives to the class of 'combinatorial problems that deserve a name', and to investigate heuristics and additional special cases that prove to have efficient solutions." Knuth and Raghunathan prove that the Metafont-labeling problem, a special case of the point-labeling problem, is NP-complete.

In the artificial intelligence (AI) community, the problem of compatible representatives has been addressed as the *constraint satisfaction problem* (CSP). A CSP consists of a finite set V of variables (corresponding to our features), of finite variable domains, i.e. sets D_v of at most d values (our label candidates) for each variable v in V, and of relations R on subsets V_R of V that exclude certain combinations of values for V_R. If we use symmetric binary relations that exclude intersecting candidates for each pair of features, the label-placement decision problem fits into this framework. The usual objective in the AI community

is either to list *all* assignment tuples without conflicts [MF85], to minimize the number of conflicts [FW92], or to find the maximum weighted subset of constraints that still allows an assignment (Max-CSP) [SFV95]. Since graph coloring and the decision version of the label-placement problem are NP-hard special cases of CSPs, one cannot expect to solve general CSPs in polynomial time. For this reason, the class of *network-consistency algorithms* has been invented. These algorithms use local arguments to exclude values from the domain of a variable that cannot be part of a global solution. Network-consistency algorithms can be seen as a preprocessing step to backtracking since they often reduce the search space very effectively.

In Chapter 2, we introduce a new framework for the general label-placement problem. We first extend classical CSP in order to be able to express the label-number maximization problem within this new framework. Then we develop a new form of local consistency, namely *r-irreducibility*. We present an algorithm, EI-1, that achieves 2-irreducibility in $O(d^3e)$ time using $O(de)$ space, where d is the size of the variable domains and e the number of binary relations. We also give a simple algorithm that finds near-optimal solutions for problems within our framework by combining EI-1 with a heuristic. This algorithm, EI-1* has proven to perform very well in practice, see Section 3.2, where we apply it to the point-labeling problem.

The following chapters are devoted to special cases of the general label-placement problem. In Chapters 3 to 5, we investigate the problems of labeling point and line features. When labeling a set of points, two fundamental questions can be asked. First, how many points can be labeled and second, how large can the labels be if all points must be labeled.

1.6.2 Point Labeling: Label-Number Maximization

In Chapter 3, we focus on label-number maximization given axis-parallel rectangular label candidates. First, we present two classes of models for labeling points with axis-parallel rectangles, namely so-called *fixed-position* and *slider* models. While the former restrict the number of candidates to a constant, the latter allow an infinite number. We compare some of these models theoretically by showing how many more points can be labeled in one model than in another. This is joint work with Marc van Kreveld and Tycho Strijk, both at Universiteit Utrecht [vKSW98, vKSW99].

Next, we exemplify our general framework at one of the fixed-position models. We do this such that it becomes clear how our concept can be applied to other cases. The resulting algorithm is fast, simple and performs well even on large real-world data sets. We study competing algorithms and do a thorough empirical comparison. It turns out that our algorithm produces results comparable to simulated annealing but obtains them much faster. Our algorithm outperforms a heuristic of Kakoulis and Tollis [KT98], not only in terms of time, but also in terms of quality. Like our framework, both simulated annealing [ECMS97] and the heuristic of Kakoulis and Tollis can be applied to the

general label-placement problem.

Since our framework is limited to fixed-position models, we also propose a fast greedy algorithm that works for slider and fixed-position models. Then we show that the slider models have polynomial-time approximation schemes. Finally we compare the greedy algorithms for slider models experimentally to those for fixed-position models. This part is also joint work with Marc van Kreveld and Tycho Strijk [vKSW98, vKSW99].

1.6.3 Point Labeling: Label-Size Maximization

In Chapter 4, we look at the second aspect of point labeling, namely label-size maximization. Instead of asking how many features can be labeled given candidates of a fixed size, we now assume that all points must be labeled and that their labels all have the same size. Under these circumstances it is natural to search for algorithms that simultaneously maximize the size of all labels. In the case of square label candidates, four per point, a theoretically optimal algorithm is known [FW91, Wag94] and has been extended to perform very well in practice [WW97]. We propose an algorithm for labeling points with uniform circles. The algorithm guarantees to find a placement with circles of about 1/20 of the diameter of the labels in an optimum solution. This improves the only known algorithm [DMM$^+$97] by more than 50%. We also show that it is NP-hard to approximate the problem beyond a certain constant factor. This is joint work with Tycho Strijk.

1.6.4 Line Labeling

While an abundance of solutions for point labeling and some acceptable approaches to area labeling have been suggested, mostly using the medial axis [AF84] or methods for computing the largest enclosed rectangle of given aspect ratio [vR89, AIK89, CK89, DMR97], there seems to be a gap in the literature concerning efficient geometric algorithms for labeling linear features such as rivers or streets. In Chapter 5 we turn our attention to line labeling. There the emphasis does not lie on maximizing label number or size, but on the question where to place the label in the vicinity of the object to be labeled. In other words, we are confronted with a modeling rather than an optimization problem. We first list the requirements of high-quality line labeling and divide them into two categories, *hard* and *soft* constraints.

In Section 5.3, we propose an efficient algorithm that produces a candidate strip along the input polyline. The strip has the same height as the given label, consists of rectangular and annular segments, and guarantees the hard constraints, such as a lower bound on a label's distance to the polyline and on the label's curvature.

In Section 5.4, we present algorithms for several evaluation functions whose task is to produce one or several good label placements within the candidate strip. These functions optimize soft constraints, such as the number of inflec-

tions. Again, we perform a thorough experimental analysis by applying our algorithm to synthetic as well as real-world data, see Section 5.5.

Although several line-labeling algorithms have been proposed in the literature [Coo88, DF92, BL95, AH95, ECMS97, Kra97, Bar97, PZC98, SvK99], our algorithm is the first where at the same time curved labels are allowed and bounds on the runtime given. Chapter 5 is joint work with Lars Knipping, Freie Universität Berlin, Marc van Kreveld, Tycho Strijk, both at Utrecht Universiteit, and Pankaj K. Agarwal, Duke University [WKvK+99].

1.6.5 Designing Geometric Algorithms

In order to support the claim of the practical relevance of our concepts, we implemented most of the algorithms we propose. The experience we have gained from implementing led to a generic design concept for geometric algorithms, which we present in Chapter 6 in the form of a tutorial. Our concept greatly increases the flexibility of an implementation without sacrificing its ease-of-use. The gain in flexibility can reduce implementation effort by facilitating code reuse. Reusability in turn helps to achieve correctness since more users mean more testing. The loss in terms of efficiency is small.

Our concept is based on the *generic programming* paradigm that has evolved over the last few years. Generic programming is about making programs more flexible by making them more general [BS98]. Abstracting from concrete in- or output data representation is an example of generic programming. This paradigm has been so successful that a model—the *Standard Template Library* (STL) [MS96]—was created and added to C++, currently one of the most popular programming languages. The STL is a library of generic components, i.e. of algorithms, data containers, and *iterators* mediating between the former two. Iterators help to decouple algorithms from the type of data container they operate on. While iterators have been known before, the real novelty of the STL was the introduction of a requirements-based taxonomy of iterators, which gives a guideline for full decoupling, and an implementation of this taxonomy using the C++ template mechanism. By becoming part of the C++ standard, the STL has attracted considerable attention and has itself set a standard for good design.

After the introduction of the STL further concepts such as *data accessors* have been suggested in the C++ literature to help programmers make their implementations even more generic [Küh96, Wei97]. Data accessors are a means to further decouple an implementation from the representation of in- and output data [KW97]. So far, these extensions have been applied predominantly to graph problems [NW96]. Exceptions such as [Wei98, Ket98] deal with the representation of geometric objects, not with the design of geometric *algorithms*, our main interest here. In order to show the relevance of STL-style generic programming including later extensions as data accessors for geometric algorithms, we investigate a simple rectangle-intersection algorithm that follows the well-known sweep-line paradigm. Using this example we give a step-by step guide from an inflexible, naive interface to a truly flexible interface that sup-

ports code reuse. These steps reflect our own change of perspective during the implementation of our label-placement algorithms. We base our presentation on C++. While the ingredients of our concept have already been known, to our knowledge this is the first time that they are applied so rigorously to a geometric problem, that they are made accessible in the form of a tutorial and that they are accompanied by a thorough experimental analysis on random and real world data. Chapter 6 is joint work with Vikas Kapoor, Freie Universität Berlin, and Dietmar Kühl, Claas Solutions GmbH.

During the implementation phase of our algorithms, we developed a tool for the automatic generation and maintenance of makefiles that we found very helpful for administering inter-file dependencies in software projects [SW98]. This was joint work with Sven Schönherr, Freie Universität Berlin.