Karlsruhe Institute of Technology

Institut für Theoretische Informatik
Lehrstuhl für Algorithmik I

Consistent Labeling of Dynamic Maps Using Smooth Trajectories

Diplomarbeit
von

Benjamin Niedermann

an der Fakultät für Informatik

Erstgutachter:	Prof. Dr. Dorothea Wagner
Zweitgutachter:	Prof. Dr. Peter Sanders
Betreuende Mitarbeiter:	Dr. Martin Nöllenburg Dipl.-Inform. Andreas Gemsa

Bearbeitungszeit: 19. Dezember 2011 – 18. Juni 2012

KIT – Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

www.kit.edu

Danksagung

An erster Stelle möchte ich Dr. Martin Nöllenburg und Andreas Gemsa danken, die diese vorliegende Diplomarbeit mit großem Engagement betreut haben und die mir im letzten halben Jahr mit vielen gewinnbringenden Diskussionen und Ratschlägen zur Seite standen. Ebenso danke ich Frau Prof. Dr. Wagner für die Möglichkeit meine Diplomarbeit an ihrem Lehrstuhl zu schreiben.

Nicht zuletzt gilt ein großer Dank Max Kramer sowie meinen zwei Geschwistern Felicitas und Florian, die jeweils Teile dieser Diplomarbeit Korrektur gelesen haben. Insbesondere danke ich Max Kramer für seine vielen nützlichen Hinweise bezüglich der Verständlichkeit der Arbeit.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 18. Juni 2012 .....

Benjamin Niedermann

Zusammenfassung

Die automatische Beschriftung von Karten gewinnt zunehmend an Bedeutung. Insbesondere für Karten, die dynamisch erstellt werden oder auf die ausschließlich eine dynamische Sicht zur Verfügung steht (z.B. wie im Anwendungsfall eines Navigationsgerätes), ist eine manuelle Beschriftung nicht möglich. Daher wurde bereits viel Arbeit in die Erforschung von automatischer Beschriftung von Karten investiert. Selbst im statischen Fall, also unter der Annahme, dass sich die Orientierung, Skalierung und Position der Karte nicht ändert, fallen die meisten Formulierungen von Beschriftungsproblemen in den Bereich der $\mathcal{NP}$-Schwere. Folglich ist im Bereich der dynamischen Kartenbeschriftung nicht mit besseren Ergebnissen zu rechnen [KM03]. Gleichzeitig wurde mit diesem Ergebnis die Suche nach empirisch guten Heuristiken und beweisbar guten Approximationsalgorithmen eröffnet.

Während in [BNPW10] das spezielle Problem betrachtet wird, dass sich die Skalierung der Karte dynamisch und kontinuierlich verändert, werden in [GNR11] kontinuierlich rotierende Karten betrachtet. Aufsetzend vor allem auf der letzten der beiden Arbeiten, wird in dieser vorliegenden Diplomarbeit ein ähnliches Problem angegangen: Unter der Annahme, dass wir auf eine gegebene Karte nur mit eingeschränktem Sichtfenster blicken, untersuchen wir den Fall, dass wir für dieses Sichtfenster eine beliebige aber fest vorgegebene Trajektorie betrachten. Es stellt sich dann die Frage, inwiefern eine optimale Beschriftung ohne Überlappung von Beschriftungselementen der Karte gefunden werden kann, wenn sich dieses Sichtfenster bezüglich dieser Trajektorie ausrichtet und die sichtbaren Beschrifungselemente sich zu Zwecken der besseren Lesbarkeit in ihrer Ausrichtung dem Sichtfenster anpassen. Diese Problemstellung leitet sich aus dem Anwendungsfall ab, dass man den Verlauf einer vorgegebenen Route auf einer Karte verfolgt (z.B. mithilfe eines Navigationsgerätes oder Smartphones).

Für die Diskussion dieser Problemstellung ist die Arbeit in zwei Teile gegliedert: Im ersten Teil der Arbeit gehen wir näher darauf ein, wie eine solche Trajektorie modelliert werden kann. Hierzu nehmen wir an, dass bereits ein Polygonzug vorgegebenen ist (eine übliche Modellierung von Routen), der dann in eine glatte Kurve überführt werden soll. Um eine diskretisierte Beschreibung dieser Kurve zu erhalten, setzen wir diese aus endlich vielen Kreisbögen zusammen. Hauptuntersuchungsgegenstand in diesem Teil der Arbeit ist die Frage, wie ein Polygonzug P möglichst gut von einer glatten Kurve K approximiert werden kann, so dass K kleine Schlingen und Ausreißer von P ausgleicht und dennoch in einem vorgegebenen Bereich um P liegt.

Im zweiten Teil der Arbeit betrachten wir dann das eigentliche Beschriftungsproblem: Zuerst gehen wir der Frage nach, wie ein entsprechendes Modell formuliert werden kann. Aufbauend auf diesem Modell beschreiben wir dann Methoden, mit deren Hilfe man bestimmen kann, wann sich zwei Beschrifungselemente überlappen und für welchen Zeitraum diese Überlappung sichtbar ist. Hinzu werden wir zeigen, dass auch dieses Beschriftungsproblem $\mathcal{NP}$-schwer ist, so dass wir uns folglich auf die Suche nach entsprechenden Lösungen machen. Präsentieren können wir schnelle Heuristiken und exakte ILP-Formulierungen. Um dieses Heuristiken auch testen zu können, wurden große Teile der Arbeit bereits in die Praxis umgesetzt, so dass wir empirische Ergebnisse auf Basis von Realweltdaten vorstellen können.

Contents

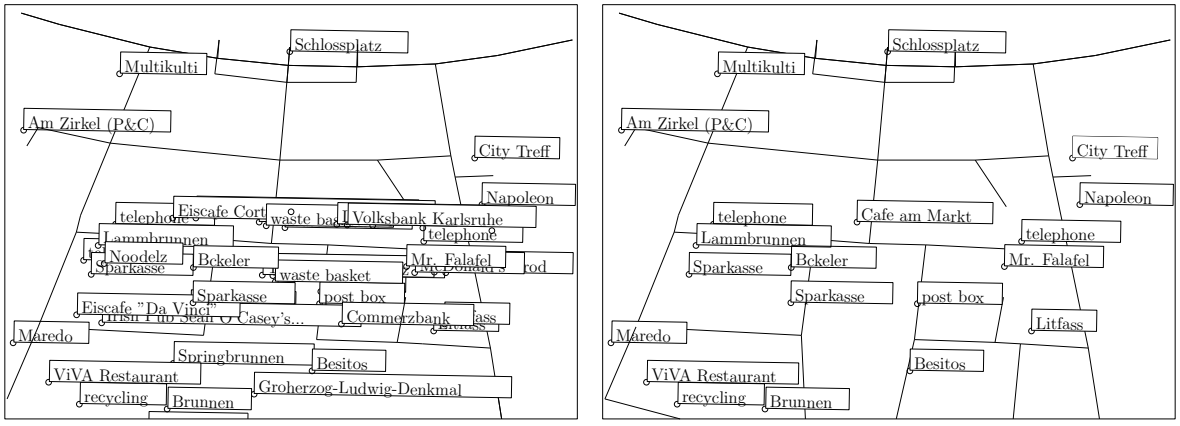
1. Introduction

With an increasing number of portable devices as navigation systems and smartphones, digital maps play a more and more important role in daily life. Many people use these systems everyday for finding the closest parking garage, a nearby restaurant or just the closest café. On the other hand, the information that can be possibly depicted on a map increases unremittingly: Apart from places of public interest as hospitals, libraries, and parks, every shop, every gas station and every pub wants to show up on the most popular maps for portable devices in order to attract customers.

Thus, it is all the more important to offer the user of such a device a clear non-distracting view on the most important features of a map such that the map and the labels describing the features of the map are still legible. Especially in the case that those devices are used for purposes of navigation in road traffic, it is crucial for users of such a device that they can obtain all relevant information of the considered map at a glance without endangering their safety. Thus, the information of a map must be filtered first, before it can be presented. Figure 1.1 illustrates the comparison of a map with and without filtering labels.

Great effort has already been made for finding appropriate approaches for filtering and drawing labels clearly on a map[1]. First static maps have been considered, that is, maps that do not change their scale, alignment or visible section, whereat we also call the latter one the *viewport* of the map. Then a typical problem to be solved is called the *label number maximization problem*: How can a maximum cardinality subset S of the given labels be found such that all bounding boxes of the labels in S can be drawn without overlaps?

Moreover, further requirements must often be satisfied. For example a label should be placed closely to the location that is described by that label or a certain point of that label



(a) Without label filter.

(b) With label filter.

Figure 1.1: Shows the center of Karlsruhe without and with using a label filter based on a scale of 1:2000.

[1]For a summary of that work see Chapter 13 – Related Work

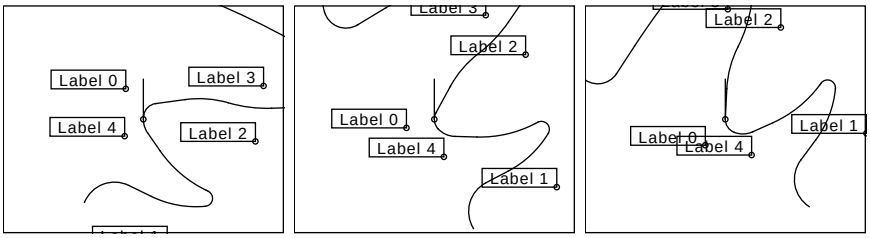


Figure 1.2: Moving along a trajectory. The viewport changes over time such that the center of the area moves along the trajectory and the axes of the area is adjusted to the direction of the trajectory.

must coincide with that location. Anticipating, most of the labeling problems formulated in such a way are $\mathcal{NP}$-complete. Consequently, one often tries to find empirical good heuristics and algorithms with provable approximation guarantee solving the labeling problem.

Dynamic maps bring the problem to the next stage: Now the map can also be changed regarding its scale and alignment over time. Less research has been done for the *dynamic label number maximization problem* than for the static case. One has to think about how conflicts between labels can be resolved dynamically. To that end one first has to define an appropriate behavior for labels such that distracting effects for labels as sudden changes of position and size or frequent appearance and disappearance are omitted.

Most of the research has been done for the case that the observer of the map can interact with that map freely[1], e.g., continuously changing the zoom, rotation or viewport of the map. However, for a lot of use cases, as for example navigation systems, the viewport of the map is pre-defined, as one considers a pre-computed route that one wants to follow. In that case it is likely that one can use that a priori information in order to gain solutions for the dynamic labeling.

In this thesis we exactly consider the case that the viewport of the map changes over time by following a pre-defined trajectory, as if we look at the map through the lens of a camera moving continuously along the smooth trajectory (see Figure 1.2). Moreover, the orientation of the viewport is adjusted to the direction of the trajectory such that moving along a curve means that the viewport begins to rotate relative to the map. In order to be able to compute the exact location and alignment of the viewport, we model trajectories by means of a chain consisting of circular arcs. Inspired by polylines, we call that chain *polyarc*. Instantly, the question arises how one can gain such a polyarc. We will introduce several approaches that translate polygonal chains into polyarcs assuming that a polygonal chain models a route within a street map.

Apart from trajectories, we assume that a set $L = \{l_1, \ldots, l_n\}$ of labels is given that is related to a set of points $P = \{p_1, \ldots, p_n\}$ on the map such that p_i is the rotation point for the label l_i for all i with $1 \leq i \leq n$. In order to sustain the legibility of the labels we also require that the labels are axis-aligned to the viewport. Consequently, the labels also rotate when the viewport rotates, which may have the effect that visible labels overlap for a certain period of time (see Label 0 and Label 4 in Figure 1.2).

We are then going to answer the question how labels can be turned on and off in such a way that an optimal number of labels is visible without overlaps, whereat we still have to define what *optimal* means in that context. Further, we also have to think about the behavior of a label, because we do not want the label to disappear and appear in a row such that it has distracting effects on the observer of the map. We call that labeling problem *the trajectory*

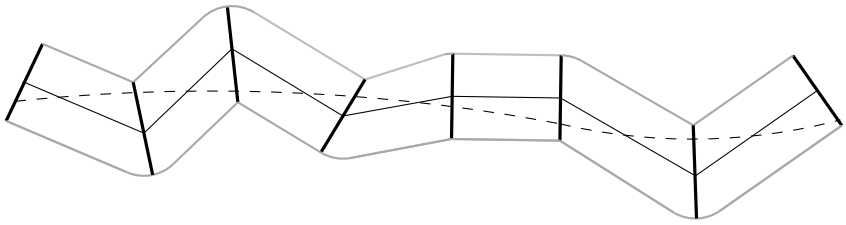


Figure 1.3: A polygonal chain (black) with tolerance region (gray) approximated by a polyarc (dashed curve). The gates of the polygonal chain are bold.

based labeling problem (TBLP).

In order to answer the arising questions, we divide this thesis into two parts. While the first part explains how polygonal chains can be translated into polyarcs, in the second part we analyze how TBLP can be modeled an solved based on polyarcs. Even though, the second part uses the first part, both can be read independently from each other, as we use the first part as black-box for the second part. In the following we summarize shortly their structure and the main results we have achieved.

1. Part – Trajectories Consisting of Circular Arcs: In that part we explain how a smooth trajectory can be obtained. As the main use case are navigation systems, we assume that we are given a polygonal chain P, which is a common way to model routes within a street map. We then explain how P can be transformed into a smooth curve K based on circular arcs such that K approximates P appropriately, that is, it compensates little oscillations and outliers of P still approximating its shape within a certain tolerance region.

To that end we follow two approaches: The first one is based on the idea that we translate first P into a polygonal chain P' approximating P such that P' has minimal number of segments and inflection points. We then translate P' into a polyarc by replacing its kinks with circular arcs. Within this step we preserve the number of inflection points, however, we have to introduce further segments so that we lose the optimality regarding the number of segments. In particular for translating P into P', we introduce an approach that needs $\mathcal{O}(n)$ time, when n denotes the size of P. As we also need $\mathcal{O}(n)$ time for resolving the kinks of P' by means of circular arcs, we obtain a procedure for translating P into a polyarc that needs $\mathcal{O}(n)$ time in total.

The second approach is based on the idea that we directly find a polyarc approximating P. For that purpose we introduce a tolerance region around P within the polyarc must lie. Further, for each vertex p of P we define a line segment through p such that it divides the tolerance region into two parts (see Figure 1.3). We call those line segments the *gates* of P. Then we want to find a polyarc of minimal number of segments such that the transition of two segments lies on a gate.

In order to solve the problem, we use circle shooting, a generalization of ray shooting: The idea is that we translate the border of the tolerance region into obstacles that must not be clashed by the polyarc we are looking for. We then compute for each gate g of P the last gate reachable from g by *shooting* circular arcs to the consecutive gates such that they do not clash with obstacles. To that end we generalize circle shooting: So far only approaches are known that shoot a circular arc from a fixed point p to line segments in order to determine the information whether the line segments are reachable from p. Our algorithm computes not only for a fixed point but for a fixed line segment whether another line segment is reachable.

The time complexity of that algorithm lies within $\mathcal{O}(m)$ time when m denotes the number of obstacles.

We then use this approach in order obtain an algorithm using $\mathcal{O}(n^4)$ time that computes a not necessarily smooth polyarc with minimal number of segments as described above. This algorithm can be seen as a generalization of the approach described in [DRS08]. In order to improve the running time, we introduce two variants of an approximative algorithm using $\mathcal{O}(n^2)$ time that returns a polyarc K lying within the tolerance region of P as follows:

1. K is not necessarily smooth and at most twice as long as the optimal solution.
2. K is smooth and at most three times as long as the optimal solution.

2. Part – Consistent Labeling Based on Trajectories: In that part of the thesis we analyze the trajectory based labeling problem. For that purpose we use the first part as a black-box and assume that we are given a trajectory based on polyarcs. We then first adapt already known models [BDY06, GNR11, BNPW10] for the labeling problem in order to formalize TBLP. In particular we solve the problem how the visibility of labels and the visibility of overlaps of labels within a given viewport can be obtained and related to the given trajectory.

Based on that model, we present a formal definition of TBLP requiring a certain behavior of the labels: We follow the idea that a label may only be turned on or off within the viewport, if there exists a direct reason why the state of the label is changed, e.g., the label begins or end to overlap with another label. We use that idea to formulate three different variants of TBLP requiring different label behavior.

In the model definition we assume that the duration of the visibility of a label is measured by the length of the part of the trajectory for which the label is visible within the viewport. We also discuss the alternative that comprises the area of the label that is visible within the viewport. We give a detailed description how that visibility can be obtained.

Further, we prove that TBLP is $\mathcal{NP}$-complete and present integer linear programs (ILP) for the different variants of TBLP. Due to the $\mathcal{NP}$-completeness of TBLP we give some heuristics for solving TBLP and compare them to the ILPs by means of an implementation of this part of the thesis using real world data. This implementation not only comprises an automatic mode for benchmarks, but also a mode for interactive use.

General Notes for Reading this Thesis: Before we start with the first part of the thesis, we present in Chapter 2 foundations that are necessary for reading the thesis. Further, each of both parts contain an own introduction explaining the considered problem more precisely and formally than it has been done so far. These introductions also describe the structure of the part they belong to. In order to close the single parts, each part also contains a chapter at its end summarizing the result we have achieved.

2. Preliminaries

In this chapter we explain some basic terms that we need for both parts, as for example curves, circular arcs and polyarcs. In the two main parts of this thesis, we then extend and adapt those terms from to time.

Curve: We define a *curve* c as a function $c : [a, b] \to \mathbb{R}^2$ where $-\infty \leq a \leq b \leq \infty$ (see Figure 2.1). We require that except at a and b the curve c is differentiable. In the case that a or b is not restricted we say that c is *infinite* otherwise *finite*. In the case that a is restricted we call a *the starting point of* c. We then assume without loss of generality that $a = 0$. Analogously, if b is restricted we say that b is *the end point of* c. We then assume with out loss of generality that $b = 1$. Consequently a finite curve c is defined on the interval $[0, 1]$. Further, we call a number $r \in [a, b]$ *a position on or of* c. For a finite curve c with starting point p ($c(0) = p$) and end point q ($c(1) = q$) we also often write $c = (p, q)$.

The direction of a curve c at a position r of c is defined as the derivative of c at r. For the positions a and b we consider the one-sided limits.

Independently from whether a curve c is finite or infinite we assume that c has a pre-defined orientation: For two different points p_1 and p_2 on c we say that p_1 *lies before* p_2 if there exist positions r_1 and r_2 on c such that $p_1 = c(r_1)$, $p_2 = c(r_2)$ and $r_1 < r_2$. Analogously, we say that p_1 *lies after* p_2 if there exist positions r_1 and r_2 on c such that $p_1 = c(r_1)$, $p_2 = c(r_2)$ and $r_1 > r_2$. Thus, the starting point of a finite curve c lies before the end point of c occurs. Note that in the case in which c intersects itself a point p_1 can lie before and after a point p_2. The orientation of a curve also induces the terms *left* and *right*.

A curve c intersects itself if there are two positions r_1 and r_2 with $r_1 \neq r_2$ on c such that $c(r_1) = c(r_2)$. We call a curve c *a non-self-intersecting-curve* if it does not intersect itself. A curve $c = (p, q)$ *intersects or crosses* another curve $c' = (p', q')$ at a point s if s is a point of both curves, $s \notin \{p, q, p', q'\}$ and both curves switch sides at s. We also say that c and c' *intersect* if there is at least one common point where both curves intersect each other. If c and c' have a point s in common but they do not intersect each other at s we say that they *touch each other at* s.

For a finite directed curve $c = (p, q)$ from p to q we call the semi-line $\overleftarrow{c}$ which starts at p having the opposite direction as c has at p *the backward directed tangential continuation of* c. Analogously, we call the semi-line $\overrightarrow{c}$ which starts at q having the same direction as c has at q

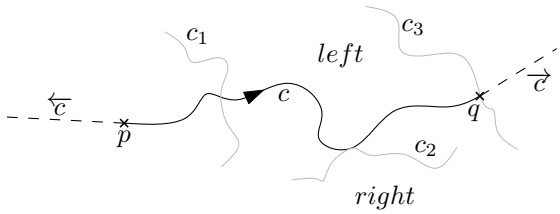


Figure 2.1: Illustration of a curve $c = (p, q)$. The curve c_1 intersects c while the other two curves c_2 and c_3 touch c.

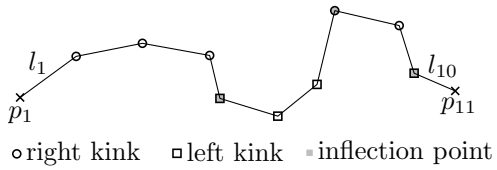


Figure 2.2: Illustration of a polygonal chain. The illustrated polygonal chain first has three right kinks, then three left kinks and finally two right kinks and one left kink.

the forward directed tangential continuation of c. Then we call the curve e assembled by $\overleftarrow{c}$, c and $\overrightarrow{c}$ having the same orientation as c *the extension of c.*

On the geometric level we can imagine a curve c as a trajectory in the plane.

Lines: On the analytic level we see a line l as a curve with constant derivative and assume implicitly that l is oriented. On the geometric level a line l is defined as usual by two points p and q. In the finite case we call $l = (p, q)$ a *line segment* which we also denote by $\overline{pq}$.

Polygonal Chain: A polygonal chain P is a sequence of points $p_1, \ldots, p_n$ in the plane (see Figure 2.2). We often identify the consecutively connecting line segments $l_1 = \overline{p_1p_2}, \ldots, l_{n-1} = \overline{p_{n-1}p_n}$ with P and also write $P = (l_1 = \overline{p_1p_2}, \ldots, l_{n-1} = \overline{p_{n-1}p_n})$. We call p_1 *the starting point of P* and p_n *the end point of P*. Further, we call the intermediate points $p_2, \ldots, p_{n-1}$ the *kinks* of P. More precisely a kink p_i is *a left kink* if p_{i+1} lies to the left of the extension of l_{i-1} or on it. Analogously, a kink p_i is *a right kink* if p_{i+1} lies to the right of the extension of l_{i-1} or on it. We call a kink p_i *an inflection point of P* if the previous point p_{i-1} is a kink of the opposite type.

Circular Arc: A circular arc is a segment of a circle. On the geometric level we therefore define a circular arc A by three points p_1, p_2 and p_3, where p_1 is the starting point and p_3 is the end point (see Figure 2.3) and say that A goes from p_1 to p_3. We also write $A = (p_1, p_2, p_3)$. If p_2 is not specified yet, we write $A = (p_1, \cdot, p_3)$, that is A is an arbitrary arc starting at p_1 and ending at p_3. From this direction we derive the terms *left* and *right*. Going the arc along from p_1 to p_3, we also use terms describing a chronological order (e.g. first the arc touches a left obstacle, then a right obstacle).

For a clockwise oriented circular arc A we also write $\overrightarrow{A}$, and call it more short *clockwise arc* (CA). Analogously, for a counterclockwise oriented circular arc A we also write $\overleftarrow{A}$, and call it more short *counterclockwise arc* (CCA).

The *corresponding circle* of a circular arc is the circle that has the identical center and radius as A.

In order to have a flowing transition between clockwise arcs and counterclockwise arcs we also describe an arc $A = (p, \cdot, q)$ by its two end points and by its *outgoing direction* $\Phi(A) \in [-\pi, +\pi]$ that defines the direction of A at q regarding the line segment $\overline{pq}$ and the tangent of A at q (see Figure 2.3). A negative outgoing direction is identified with clockwise arcs, a positive outgoing direction is identified with counterclockwise arcs. $\Phi(A) = 0$ means that A is a line segment, in this case A is both a clockwise arc and a counterclockwise arc. Analogously, we can define the *incoming direction* which is identified with the direction of A at p.

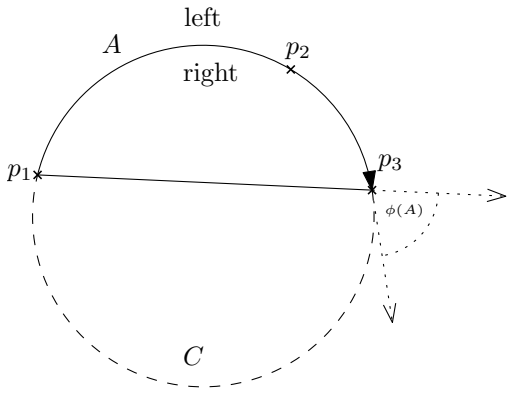


Figure 2.3: Illustration of the concept of arcs. A clockwise arc A and its corresponding circle C with a negative outgoing direction $\Phi(A)$.

Polyarc: A polyarc is a sequence $(A_1, \ldots, A_m)$ of circular arcs such that each arc A_i with $1 < i \leq m$ starts at the point where its direct predecessor ends. By definition of a circular arc, each polygonal chain is also a polyarc. In particular this means that every statement about polyarcs are also true for polygonal chains. We call a polyarc *smooth* if for all arcs A_i, A_{i+1} with $1 \leq i < m$ it is true that the outgoing direction of A_i is equal to the incoming direction of A_{i+1}.

Consecutively Disjoint: We call a sequence $O_1, \ldots, O_n$ of objects consecutively disjoint, if all consecutive pairs O_i, O_{i+1} with $1 \leq i < n$ are disjoint.

Operations on Sequences: Given two sequences $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ of arbitrary objects then we define the following operations on A and B:

$$\begin{aligned} A + B &:= (a_1, \ldots, a_n, b_1, \ldots, b_m) \\ A \cup B &:= \{a_1, \ldots, a_n\} \cup \{b_1, \ldots, b_m\} \\ A \cap B &:= \{a_1, \ldots, a_n\} \cap \{b_1, \ldots, b_m\} \end{aligned}$$

For a sequence $O_1, \ldots, O_m$ of sequences of arbitrary objects we write for $O_1 + O_2 + \ldots + O_m$ more shortly $(O_1, \ldots, O_m)$.

Part I

Trajectories Consisting of Circular Arcs

3. Introduction

In this part of the thesis we take a detailed look at polyarcs: As already mentioned in Chapter 1, in the second part of this thesis we then make use of those chains in oder to model the trajectory of a tracking shot of a camera moving over a given map. To that end we assume that we are given some polygonal chain P that describes that tracking shot. For example we could have obtained P by computing a shortest path on a street network underlaying the map. As we require that the tracking shot should not have any kinks but should be smooth, we have to think about how a polygonal chain can be transformed into a smooth curve (see Figure 3.1).

We also could have used some other concepts as splines (see [PBP02]), but circular arcs and the resulting polyarcs lend themselves to be our choice: A polyarc is assembled by a finite sequence of circular arcs that can be described by simple geometric tools. Thus, those polyarcs can be handled efficiently by geometric algorithms, such that one still can derive some guarantees.

In the following part, we describe different ways how polygonal chains can be transformed into polyarcs preserving the following properties:

1. The distance between the polygonal chain P and the corresponding polyarc K should be appropriate: Apparently, we should not transform P into an arbitrary polyarc K, but K should sustain the shape of P such that it still conveys the course of P.

2. The polyarc K should have a minimum number of arcs and inflection points: The camera should not lurch when moving over the map, but should move along P compensating small wiggly lines of P.

To that end we define the *corridor* to be the area around P that contains exactly the points having at most distance r to P, where r is a predefined number in $\mathbb{R}^+$. Then we describe different approaches that mainly sustain the following properties of K:

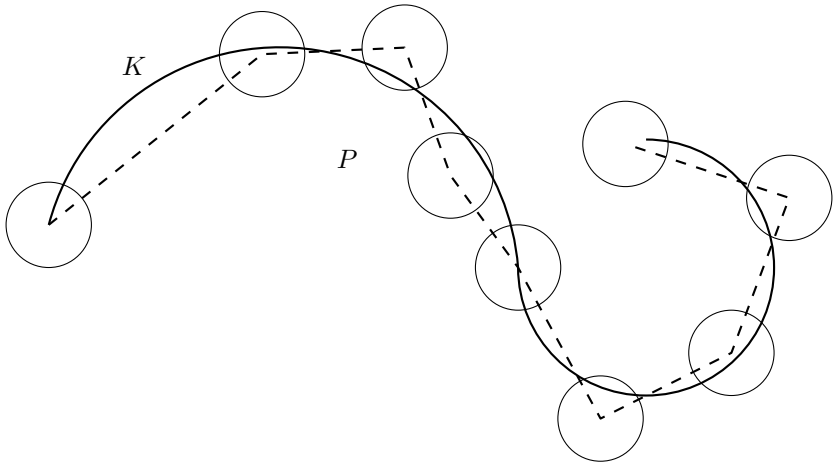


Figure 3.1: A polyarc covering a polygonal chain.

- K starts within a certain radius around the starting point of P and
- K ends within a certain radius around the end point of P and
- the end points of the circular arcs of K lie in predefined areas and
- the circular arcs of K lie in a corridor that is formed around the polygonal chain.

In order to obtain K we follow two different approaches, which both lead to several algorithms transforming P into K:

- The first approach is based on the idea that we first translate P into another polygonal chain P', such that P' satisfies certain properties, e.g., minimum number of segments or minimum number of inflection points. For that purpose we mainly use methods of line stabbing. Afterwards, we translate P' into a smooth polyarc K. Applying this step, we tolerate that some of the properties of P' are lost in order to obtain an aesthetically looking chain.

  This method is especially suited for the case that one wants to minimize the number of inflection points of K without considering the number of necessary segments.

- In the second approach we try to directly find a polyarc K covering P without the intermediate step of creating another polygonal chain. To that end we introduce a method similar to line stabbing, but using circular arcs. Due to the nature of circular arcs this is more time consuming than methods using line stabbing. Consequently, the algorithms presented for this approach are more time consuming than the ones for the first approach.

  This method lends itself for minimizing the number of segments of K. We suggest both solutions for only connected polyarcs still having kinks and solutions for smooth polyarcs. In order to reduce the time complexity we also present some algorithms that are approximative regarding the number of used segments, but faster.

We have structured the chapters as follows:

Chapter 4 – Related Work: In that chapter we shortly sketch the state of the art regarding transforming polygonal chains into polyarcs. As we make use of some of the approaches, we explain those in more detail, but still on an informal level.

Chapter 5 – Modeling the Corridor: As already mentioned in this introduction, we require the polyarc to be contained within a pre-defined corridor surrounding the given polygonal chain. In that chapter we describe how we model that corridor by geometric primitives as lines and circles.

Chapter 6 – Basic Algorithms: We describe some basic approaches how a polygonal chain can be transformed into a polyarc. While some of the approaches yield simple results, that chapter already contains more sophisticated algorithms that make use of the first main approach as presented before.

Chapter 7 – A Generalization of the Corridor: In order to introduce algorithms for the second main approach as described above, we generalize the concept of a corridor. To that end we take another view on the corridor of given polygonal chain P and describe it by means of obstacles forming borders to the left and to the right of P.

Chapter 8 – Basic Algorithms for the Generalized Corridor: Based on the previous chapter we then introduce algorithms working on the generalized corridor. First, we explain how single arcs partly crossing the corridor can be obtained, and afterwards we describe procedures how those arcs can be assembled to one polyarc such that it consists of a minimum number of arcs.

Chapter 9 – Advanced Algorithms for the Generalized Corridor: In that chapter we describe some optimizations of the previous introduced procedures and resolve drawbacks of those procedures.

Chapter 10 – Handling Special Cases: Finally, we consider some special cases that we have excluded before: For example so far we assume that the vertices of the given polygonal chain have a certain distance in order to avoid the overlapping of certain parts of the corridor. In that chapter we adapt the previous introduced algorithms for those special cases.

Chapter 11 – Conclusion: In the last chapter we summarize the results we have achieved in the previous chapters and discuss future work that can be done for transforming polygonal chains into polyarcs.

4. Related Work

Finding a polyarc K covering a given polygonal chain P is a special case of the problem of finding an approximating curve of P. For that general case splines are an well studied approach [PBP02]. However, as already mentioned in the introduction of this part, we consider polyarcs instead of splines due to their simplicity.

Covering a polygonal chain by a polyarc based on a corridor is closely related to the *visibility problem* in simple polygons: If we assume that we model the corridor as a simple polygon, one elementary question is whether one can reach a point or edge of the polygon from a fixed point by means of a circular arc contained in that polygon.

In the case that one uses lines instead of circular arcs the problem is normally called *ray shooting*. It has been extensively investigated in [CG89, AGT86]. When using circular arcs the problem is often called *circle shooting* [CW95, AS93]. Running times for solving the circle shooting problem normally lie within $\mathcal{O}(n \log n)$ time for pre-computing query structures and in $\mathcal{O}(\log^2 n)$ time for a request if n is the size of the polygon, whereat the needed time depends on how the pre-computation is done. All solutions have in common that they make the assumption that one has given a fixed point for that one wants to solve the visibility problem.

In the next section we sketch the results made by Drysdale et al. [DRS08]. They have introduced two algorithms for finding a polyarc covering P. Then in the following section we sketch the results achieved by Guibas et al. [GHMS91]. They have introduced fast line stabbing algorithms in order to cover a sequence of circles. Later on we will use those algorithms as one starting point for further algorithms.

4.1 Computing Polyarcs

In [DRS08] two algorithms are introduced for finding a polyarc K covering a given polygonal chain $P = (p_1, \ldots, p_n)$ such that K respects the order of P and lies within a given corridor which is called *tolerance region* R of P (see Figure 4.1). In order to preserve the order of P they assume that P is divided into sub-regions by so called *gates* $g_1, \ldots, g_n$. Each gate g_i is modeled by a line segment that goes through p_i. Then the idea is that K has to cross all gates in their particular order.

For the tolerance region R the authors require the following three properties:

1. "R is a simple polygon passing through all gate endpoints."
2. "R does not intersect the interior of gates or cross the segments connecting corresponding endpoints of successive gates."
3. "No line through two points on successive gates g_i and g_{i+1} crosses the portion of R connecting g_i with g_{i+1}."

It is remarkable that line segments of P which closely lie together can restrict the sub-region of the other line segment (see gray regions in Figure 4.1). For a detailed description of how to determine R they refer to [HE05].

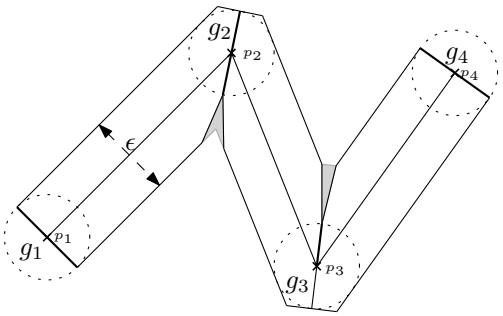


Figure 4.1: The tolerance region of a polygonal chain. Shows the tolerance region R with gates $g_1, \ldots, g_4$ surrounding the polygonal chain $p_1, \ldots, p_4$ such that the gray regions do not belong to R. The drawing is inspired by Figure 1 of [DRS08].

The two main restrictions for the polyarc K covering P is that it must completely be contained in R and that each circular arc of K must begin at a point p_i of P and end at a point p_j of P with $i < j$.

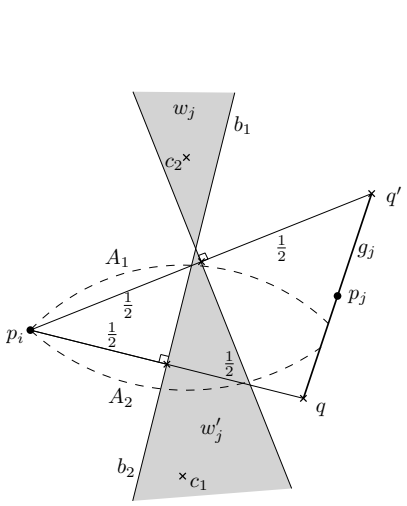
First Algorithm: The first algorithm that they introduce yields a polyarc K that is not necessarily smooth. The main idea of the algorithm is that for a point p_i and a gate g_j through a point p_j one can locate the centers of all circular arcs going through p_i and g_j in two wedges w_j and w'_j located between p_i and g_j (see Figure 4.2a): The wedges are formed by the two bisectors b_1 and b_2 between p_i and q and p_i and q', where q and q' are the end points of g_j. Based on this construction one can find for all following gates g_{i+1}, g_{i+2}, ... of p_i corresponding wedges $(w_{i+1}, w'_{i+1}), (w_{i+2}, w'_{i+2}), \ldots$ (see Figure 4.2b).

Then the intersection of those wedges form the region of all centers for arcs going from p_i through all gates that are involved in that intersection. This observation can be systematically used in order to obtain a graph encoding for each vertex p_i the reachable gates. Then based on an algorithm for the shortest path problem one can find a minimum long sequence of connected circular arcs. The authors of [DRS08] prove that this algorithm needs $\mathcal{O}(n^2 \log n)$ time.

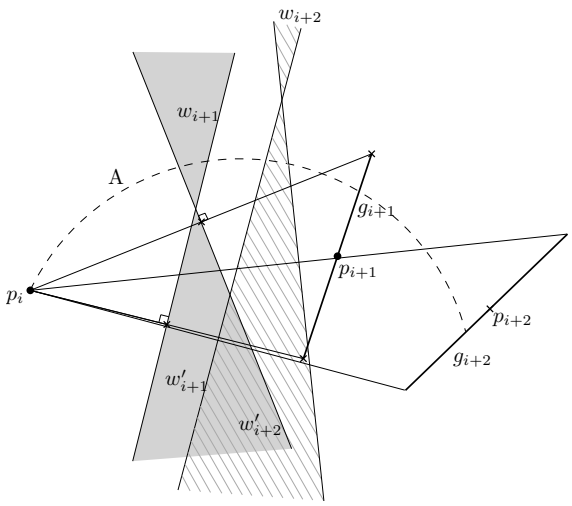
Second Algorithm: The second algorithm makes use of *biarcs*, that is, a special composition of two circular arcs [HE05, Bol75]. The authors of [DRS08] characterize a biarc B_{ij} with end points p_i and p_j as:

- "B_{ij} consists of two consecutive circular arcs a_1, a_2"
- "a_1 is an oriented arc from p_i to point p_{joint} and a_2 is an oriented arc from p_{joint} to p_j;"
- "a_1 matches the [pre-defined] tangent vector t_i at the point p_i and a_2 matches the [pre-defined] tangent vector t_j at p_j;"
- "both arcs have a common tangent at p_{joint}, the joint of the biarc."

They use the important observation that the locations of p_{joint} can be described by a circle, which they call *joint circle*. By means of circle shooting they then determine for a point p_i of P the joint circles for $p_{i+1}, p_{i+2}, \ldots$ until they reach a point p_j, such that the joint circle of p_i with p_{j+1} does not lie within R. Then they use those joint circles in order to obtain

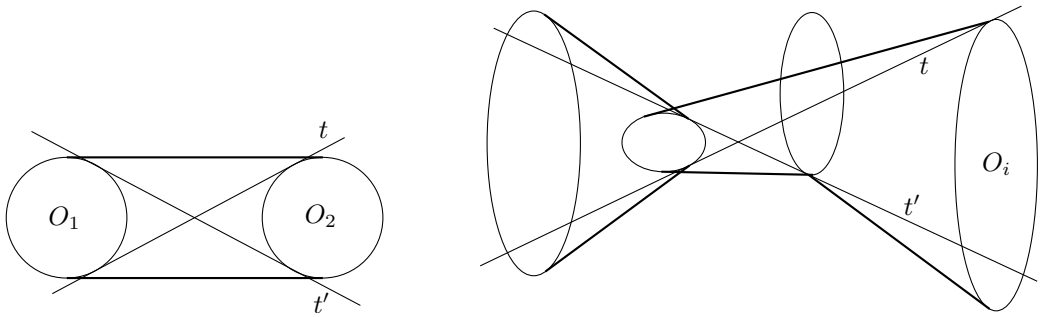


(a) The gray wedges formed by the two bisectors b_1 and b_2 of p_i and q and p_i and q' contain all centers of possible arcs connecting p_i with g_i.



(b) The gray wedges formed by the two bisectors b_1 and b_2 of p_i and q and p_i and q' contain all centers of possible arcs connecting p_i with g_i.

Figure 4.2: Region for centers.



(a) (b)

Figure 4.3: Illustration of the corridor based on tangents. Inspired by Figure 13 and 14 of [GHMS91].

biarcs lying in R. For the procedure they assume that for each point p_i in P a tangent t_i is given, which is not defined explicitly.

4.2 Line Stabbing

In [GHMS91] the authors present based on [EW91] two algorithms for stabbing a line l through a consecutively disjoint sequence $\mathcal{O} = (O_1, \ldots, O_n)$ of convex objects such that a maximum long prefix $O_1, \ldots, O_i$ is covered with respect to the order of those objects. We say that a line segment l covers a sequence $O_1, \ldots, O_i$ of objects with respect to their order, if one can find for each object O_j with $1 \leq j \leq i$ a line segment of l' that covers $O_1, \ldots, O_{j-1}$ not covering O_j. They solve the problem of finding l in $\mathcal{O}(i)$ time and space using a algorithm based on greedy methods.

Mainly they iteratively go through the sequence of objects creating successively a corridor whose existence is a witness for the existence of a line stabbing through the corresponding objects. To that end the corridor is modeled by means of outer tangents between well-chosen objects of the considered sequence such that it is guaranteed that the lines stabbing the objects must lie within this corridor. Figure 4.3a illustrates the base case in which the corridor comprises the first two objects O_1 and O_2. Figure 4.3b shows the corridor for the case that it is already extended to the object O_i.

Further, they maintain two inner tangents t and t' between objects of the already considered sequence $O_1, \ldots, O_i$ such that all lines stabbing the objects $O_1, \ldots, O_i$ lie within the wedge formed by t and t', while t and t' are obtained by means of the corridor. Then the basic observation is that the object O_i can also be stabbed, if it lies at least partly in that wedge.

On that account it is crucial to be able to compute inner and outer tangents between convex objects in order to use those algorithms. For circles it is easy to compute inner and outer tangents. For convex polygons we refer to [OvL81] which describe in more detail how one can maintain convex hulls dynamically: As a byproduct they also explain how to compute outer tangents between two convex polygon using $\mathcal{O}(m \log m)$ time and how to compute inner tangents using $\mathcal{O}(m \log^2 m)$ time, when m is the size of those polygons.

Back to [GHMS91]: They also adapt the approach as described above in order to be capable to handle circles $C_1, \ldots, C_n$ of unit size that may overlap. Since then the term of stabbing is not unique anymore, they define four different types of stabbing order, which are listed in the following:

1. l must exit the circles in the given order.
2. l must enter the circles in the given order.
3. l must exit and enter the circles in the given order.
4. l must hit pre-defined points $p_1, \ldots, p_n$ which lie in $C_1, \ldots, C_n$ in the given order.

The problem of finding a maximum long prefix $C_1, \ldots, C_i$ that can be stabbed by a line l is then solved on the base of the algorithm described before. For the first three cases they give algorithms that need $\mathcal{O}(i)$ time. For the last variant they offer a solution within $\mathcal{O}(i \log i)$ time, but all of the four variants need $\mathcal{O}(i)$ space. Later on when we make use those algorithms we consider line segments that satisfies the first or the second variant.

In the final part of their work they present an algorithm for translating a polygonal chain P into a simplified polygonal chain P' such that the line segments of P' stabs circles of given radius which can be drawn around the vertices of P. To that end they consider the corridor that arises considering those circles and their convex hull. They offer three possible locations of the vertices of P':

1. The vertices may lie anywhere.
2. The vertices must lie somewhere within the corridor of P.
3. The vertices must lie within the pre-defined circles around the vertices of P.

Depending on whether one requires that the circles are consecutively disjoint and depending on the visiting order, they give algorithms that need $\mathcal{O}(n^2 \log n)$, $\mathcal{O}(n^2 \log^2)$ and $\mathcal{O}(n)$ time. In this thesis we are mainly interested in the case that the objects are consecutively disjoint

so that one can solve that problem in $\mathcal{O}(n)$ time, respectively. Later, in Section 6.2 we suggest an alternative algorithm for that case also using $\mathcal{O}(n)$ time bringing some benefits for our purposes.

5. Modeling the Corridor

In this chapter we want to describe in more detail how we model the corridor for a given polygonal chain $P = (l_1 = \overline{p_1p_2}, l_2 = \overline{p_2p_3}, \ldots, l_{n-1} = \overline{p_{n-1}p_n})$: As already mentioned in Chapter 3, the corridor $\mathcal{C}$ of a polygonal chain consists of all points that have distance at most r to P, where r is a pre-defined positive number. In order to be able to work with $\mathcal{C}$ we describe $\mathcal{C}$ by geometrical primitives, namely line segments and circular arcs.

To that end we translate each line segment $l_i = \overline{p_ip_{i+1}}$ of P into a *corridor segment* as presented in Figure 5.1a. We call the circles C_i and C_{i+1} of radius r around p_i and p_{i+1} *connecting circles*. For the beginning we assume that all connecting circles have the same radius and that they are consecutively disjoint. In Chapter 10 we will discuss how to soften these restrictions. But up to that chapter we always assume implicitly that those circles are consecutively disjoint, if we do not state it otherwise. The border of the corridor segment is defined by the outer tangents of C_i and C_{i+1}. Since l_i is oriented we can distinguish *a left and right border* of the corridor segment.

Now we can define the corridor $\mathcal{C}$ of a polygonal chain as the sequence of the single corridor segments obtained from $l_1, \ldots, l_{n-1}$ (see Figure 5.1b). Obviously, the border of that construct encircles exactly those points which have at most distance r to P. Further, we extend the notion of *left and right borders* canonically to the corridor.

But if we only use the silhouette of the corridor for finding a polyarc K it could be that we use arcs that do not come close enough to all points of P, that is, there are some points on P which have a distance to K greater than r (see for example arc A' in Figure 5.1b). On that account we also involve the connecting circles of $\mathcal{C}$ in order to define when a circular arc lies within $\mathcal{C}$.

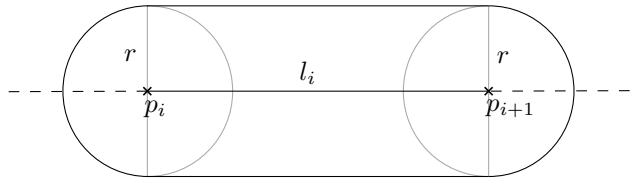
An arc A *lies in* a corridor segment if it is completely contained in the silhouette of the segment. Further, A *lies in* a given corridor $\mathcal{C}$ if we can divide A into a polyarc of sub-arcs such that each sub-arc lies within a corridor segment and each connection point of two consecutive sub-arcs lies in a connecting circle of a corridor segment. By requiring that the connection points of the sub-arcs must lie in the connecting circles we ensure that A is close enough to the end points of the line segments in P.

For example in Figure 5.1b, the arc A lies in the given corridor $\mathcal{C}$ because we can divide A into two sub-arcs such that each sub-arc is completely contained in the silhouette of a corridor segment and both sub-arcs are connected within a connecting circle. In contrast A' does not lie in C because although it lies in two corridor segments, it cannot be divided into two sub-arcs S_1 and S_2, such that the connection point of S_1 and S_2 lies in a connecting circle and each of both sub-arcs are completely contained in a corridor segment.

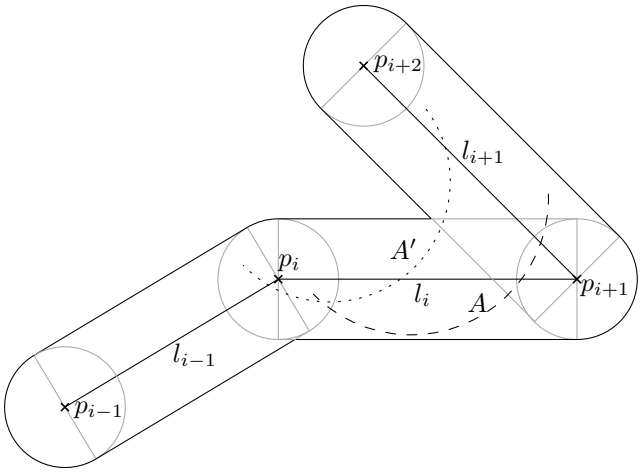
Now we want to introduce the notion of *covering* in order to be able to precisely talk about polyarcs that lie within a given corridor:

Definition 5.1. *A sequence of circles $C_1, \ldots, C_n$ is covered by a polyarc K with respect to their order if*

1. *each circle C_i ($1 \leq i \leq n$) has some points P_{C_i} in common with K and*



(a) A single corridor segment based on the line segment l_i.



(b) A corridor C for three line segments. By definition A lies in C, while A' does not lie in C.

Figure 5.1: Illustration of the concept of arcs.

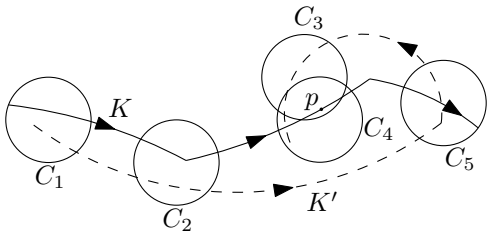


Figure 5.2: Illustration of a polyarc covering circles. While K covers all circles with respect to their order, the polyarc K' does not.

2. *for each circle C_i ($1 < i \leq n$) there is a point $p \in P_{C_i}$ such that the prefix of K that ends at p has with all circles $C_1, \ldots, C_{i-1}$ common points.*

Note that the definition is well-formed because we require that K is oriented. On that account we can speak about the prefix of K. Figure 5.2 illustrates the notion of covering circles: We also allow that K first enters C_4 before it enters C_3. In that case C_4 and C_3 must overlap and K must have at least one point p in common with the overlapping area.

We then extend this definition to the notion that K covers a corridor:

Definition 5.2. *Given a polygonal chain $P = (p_1, \ldots, p_n)$ with corridor $\mathcal{C}$ and connecting circles $C_1, \ldots, C_n$, a polyarc K* covers the corridor $\mathcal{C}$ of P, *if*

1. *all segments of K lie in $\mathcal{C}$ and*

2. *all connecting circles of $\mathcal{C}$ are covered with respect to their order.*

The first requirement mainly ensures that K is adapted to $\mathcal{C}$ closely enough, while the second one also ensures that the required order is maintained.

Finally we also want to introduce some abbreviations: We denote the corridor segment consisting of the connecting circles C_i and C_j by (C_i, C_j). We define the sub-corridor of $\mathcal{C}$ that starts with C_i and ends with C_j by $\mathcal{C}[C_i, C_j]$ (where $i < j$). Conceptually, we restrict the given polygonal chain P to the sub-chain S starting with p_i and ending with p_j. Then $\mathcal{C}[C_i, C_j]$ is equal to the corridor of S.

6. Basic Algorithms

In this chapter we present two algorithms for computing a polyarc K covering the corridor $\mathcal{C}$ of a given polygonal chain P. In Section 6.1 we present an intuitive algorithm that directly resolves the kinks of a polygonal chain in order to obtain K. In Section 6.2 we then introduce an algorithm that first simplifies P to another polygonal chain P' such that P' consists of a minimum number of line segments and inflection points still covering $\mathcal{C}$. Then, based on the previous algorithm we translate P' into a polyarc.

6.1 A Simple Algorithm for Gaining a Smooth Polyarc

In this section we introduce an algorithm using $\mathcal{O}(n)$ time that translates a given polygonal chain P into a smooth polyarc K such that K covers the corridor of P. Mainly, the algorithm follows the intuitive idea to translate left and right kinks in a combination of circular arcs. A similar concept has already been used in [FL91].

Since we normally do not want to translate every kink of P into circular arcs, but would like to have the possibility to cover several kinks by one segment of K, we assume that we also have given a polygonal chain P' that simplifies P. Then we work on P' and the corridor $\mathcal{C}$ of P in order to obtain K. The idea is that we choose P' based on P such that P' is optimal with respect to criterion, e.g., P' consists of a minimum number of inflection points. In Section 6.2 we explain in more detail how to gain P', but for the moment assume that P' is given.

Translating P' into K can yield a loss of its optimality, but we introduce a threshold by which we can control how close K is adapted to P': The closer K is adapted to P' the smaller the circular arcs are. On that account we still give the user of the algorithm the decision between optimality and quality of K.

Of course we have to require properties from P' so that we can always translate it into K such that K respects the corridor of P. In order to be not too restrictive P' must only satisfy the following property:

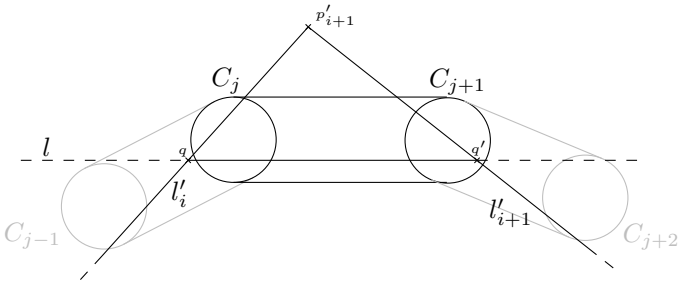
Property 6.1. *P' covers the connecting circles of $\mathcal{C}$ such that each line segment of P' covers at least two circles.*

We now explain how one can translate two consecutive line segments of P' into a polyarc that we can directly use for K. For that purpose we assume that P is given as $P = (p_1, \ldots, p_n)$ and P' is given as $P' = (l'_1 = \overline{p'_1 p'_2}, \ldots, l'_{m-1} = \overline{p'_{m-1} p'_m})$. Let $C_1, \ldots, C_n$ be the sequence of connecting circles of $\mathcal{C}$, then we know for two l'_i and l'_{i+1} in P' that l'_i and l'_{i+1} meet around a corridor segment (C_j, C_{j+1}), that is, l'_i covers at least C_j of both circles and l'_{i+1} covers at least C_{j+1} of both circles (see Figure 6.1).

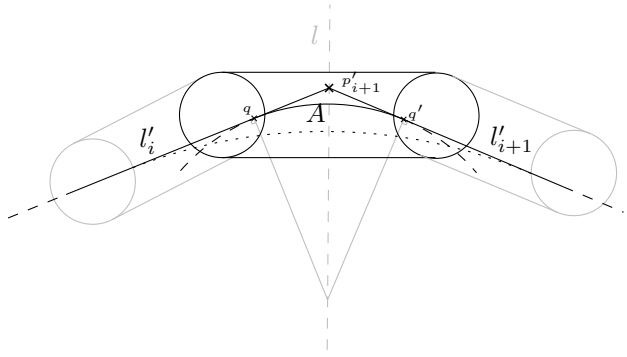
We divide the algorithm into two steps:

1. For two consecutive line segments l'_i and l'_{i+1} we check whether the point p'_{i+1} lies outside (C_j, C_{j+1}). If that is the case we replace that kink with two other kinks having the same orientation.

2. We resolve the kinks introducing arcs bridging those kinks.



(a) The right kink lies outside the corridor segment.



(b) The right kink lies inside the corridor segment.

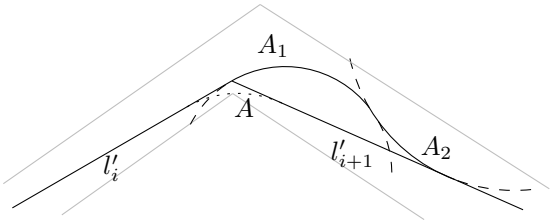
Figure 6.1: Resolving right kinks.

We now explain both steps in detail. Since left and right kinks can be handled analogously, we only consider right kinks explaining both steps.

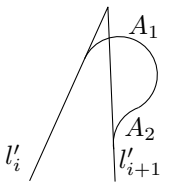
First Step: We go through all line segments of P' and check for two consecutive line segments l'_i and l'_{i+1} whether p'_{i+1} lies outside the corridor segment (C_j, C_{j+1}) (as defined above). As already mentioned we assume that p'_{i+1} is a right kink. If p'_{i+1} lies outside of (C_j, C_{j+1}), we can conclude that p'_{i+1} must lie to the left of (C_j, C_{j+1}), because otherwise l'_i could not cover C_j, while l'_{i+1} is covering C_{j+1}. We resolve the right kink at p'_{i+1} by introducing two new right kinks, that lie within (C_{j-1}, C_j), (C_j, C_{j+1}) or (C_{j+1}, C_{j+2}). To that end we draw a line l through (C_j, C_{j+1}) such that l is parallel to the borders of (C_j, C_{j+1}) (see Figure 6.1a). Since both l'_i and l'_{i+1} intersect the left border of (C_j, C_{j+1}) and both lines cover at least two circles, we know that l intersects both lines in points q and q' forming right kinks such that q and q' lie within (C_{j-1}, C_j), (C_j, C_{j+1}) or (C_{j+1}, C_{j+2}). Conceptually, we then replace p'_{i+1} with q and q' in P'.

We do not explain which of the possible lines going through the corridor one should take. It is likely that one want to take a line that lies to the left of $\overline{p_j p_{j+1}}$ connecting the centers of C_j and C_{j+1}, because then one increases the distance between q and the right border of (C_j, C_{j+1}) and the distance between q' and the right border of (C_j, C_{j+1}). Why this can be useful is explained after we have described the second step.

Second Step: After applying the previous step, we now can assume that all kinks lie within the corridor. Again we iterate through all line segments of P' including the new ones added in the first step. For two consecutive lines l'_i and l'_{i+1} we resolve the kinks as follows (again explaining only right kinks): Let l be the bisector of l'_i and l'_{i+1}, then we draw an arc A such



(a) Alternative for the case that the A (dotted) has a too small radius.



(b) Alternative for the case that the angle between l'_i and l'_{i+1} is too small.

Figure 6.2: Alternative constructions.

that its center lies on l and A touches l'_i and l'_{i+1} in points q respective q' (see Figure 6.1b). Further, we require that A lies within in the corresponding corridors and that q lies on the second half of l'_i and q' lies on the first half of l'_{i+1}. The two last requirements guarantee that we obtain a smooth chain when apply the same approach on the previous and the following kink.

We again leave the question be unanswered how to choose the arcs exactly. For aesthetic reasons we suggest maximizing the radius of the arcs, such that they cover the corridor as much as possible. Analyzing both steps, we can state that none of the steps increases the number of inflection points in P', because we only consider one kink type.

However, K can have more segments than the original polygonal chain P' has: Since in the first step we introduce for each kink at most two vertices (q and q') but delete one (p'_{i+1}), we obtain at most $2m$ vertices after the first step, if m is the number of vertices in P'. In the second step we introduce for each intermediate vertex one arc, such that we introduce at most $2m-2$ vertices in total in the second step. Summarizing we obtain at most $4m-2$ vertices. Since due to Property 6.1 it is true that $m \leq n$ we also can conclude that K has at most $4n-2$ vertices, where n is the size of P. The following theorem sums up the results:

Theorem 6.1. *Given two polygonal chains P and P' such that P' satisfy Property 6.1. Let n be the number of vertices in P, then we can obtain a smooth polyarc K such that*

1. *K has at most $4n-2$ vertices and*
2. *K has the same number of inflection points as P'*

in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.

Alternative Constructions: Depending on P' it is possible that the arcs in K are shorter than desired or have a small radius. We show how one can solve this problem based on thresholds. Again, we only discuss right kinks, since left kinks can be handled analogously:

We observe that if l'_i and l'_{i+1} meet in a right kink at the point p'_{i+1}, then the radius of the arc A bridging p'_{i+1} becomes smaller the closer p'_{i+1} lies to the right border of the corridor. We suggest a solution based on a threshold r_{min} stating the smallest allowed radius: If all possible arcs bridging p'_{i+1} fall short of r_{min}, then we introduce the solution as depicted in Figure 6.2a: We smoothly continue the line segment l'_i by an arc A_1 and let A_1 fade to another arc A_2 touching l'_{i+1}.

Analogously, we introduce alternative constructions for the case that the angle between l'_i and l'_{i+1} is smaller then a pre-defined threshold α_{min}(see Figure 6.2b).

Note that both corrections are only suggestions and that there are many other alternative constructions. In order to keep the number of newly introduced segments and inflection points small, one should think about constructions that consist of few segments and inflection points. Although we lose the optimality regarding P', such constructions avoid too small radii and angles. For the presented alternative constructions the next lemma follows directly:

Lemma 6.1. *Using the presented alternative constructions K has at most twice as many inflection points as P'.*

6.2 A Polyarc with Minimum Number of Inflection Points

Based on the section before, we introduce an approach that yields for a given polygonal chain $P = (p_1, \ldots, p_n)$ a smooth polyarc K, such that K has a minimum number of inflection points. Let $C_1, \ldots, C_n$ be the connecting circles of the corridor $\mathcal{C}$ of P, then the main idea is to find a polygonal chain P' such that

1. P' covers $\mathcal{C}$ satisfying Property 6.1 (see Page 27) and
2. P' has a minimum number of inflection points.

Since P' satisfying Property 6.1, we then can apply the approach of Section 6.1 in order to obtain K. Depending on whether we also use alternative constructions in order to compensate small angles and radii we either get a solution where K has the same number of inflection points as P' or at most twice as many.

We therefore focus on the question how to obtain P'. Mainly we try to find a minimum number of line segments $l_1, \ldots, l_m$ covering the circles $C_1, \ldots, C_n$ with respect to their order. In [GHMS91] the authors suggest a similar method based on line stabbing, which also results a polygonal chain P' that covers all circles (see Section 4.2). We present the following algorithm as an alternative which persuades with its simplicity. Further, it allows to plug in different methods for covering circles by one line. For example for circles of the same radius there are fast algorithms for line stabbing, while for circles of different sizes that may overlap the algorithms become slower (see [GHMS91] and [EW91]).

Besides, introducing our algorithm goes along with introducing terminology that are helpful for proving that P' has a minimum number of inflection points, which is not proven in [GHMS91]. The main concept of this section is presented in the following definition:

Definition 6.1. *Given a sequence $S = (C_1, \ldots, C_n)$ of circles, then we call a line b a* bridge *with* anchors *C_i, C_j and C_k ($1 \leq i \leq j < k \leq n$) if*

1. *b covers the sequence $C_i, \ldots, C_j, \ldots, C_k$ with respect to their order, and*
2. *C_k is the last circle in S that can be covered by a line that also covers C_j, and*
3. *C_i is the first circle in S that can be covered by a line that also covers C_j and C_k.*

Figure 6.3 illustrates the presented definition: The idea of a bridge is starting from a circle C_j we want to stab as many successors of C_j as possible and then based on those circles we want to cover as many predecessors of C_j as possible. Note that C_i and C_j are not necessarily different circles. This can happen, if going backwards from C_k the circle C_j is the last circle that can be covered by the same line. Further, it follows directly from the definition

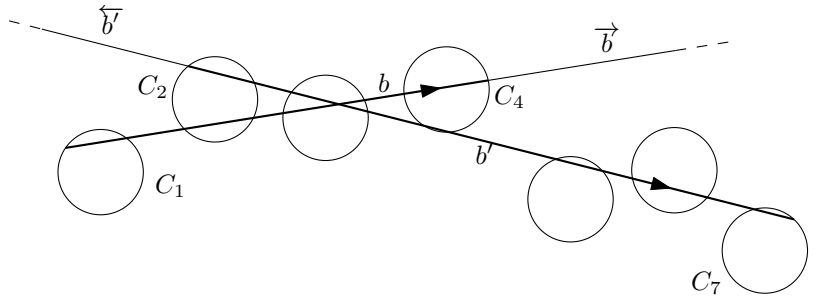


Figure 6.3: Illustration of bridges. The figure shows the bridge (b, C_1, C_1, C_4) and the bridge (b', C_2, C_4, C_7).

that all bridges b with the same intermediate anchor C_j cover also the same first anchor C_i and the same last anchor C_k .

Notations: We also often call (b, C_i, C_j, C_k) *bridge* for a bridge b with anchors C_i, C_j and C_k and see b as an oriented line going from C_i (*first anchor*) through C_j (*intermediate anchor*) to C_k (*last anchor*). Moreover, the semi-line that begins at the starting point of b and extends b to its forward direction is denoted by $\overrightarrow{b}$ and is called the *forward directed extension* of b. Analogously, the semi-line that begins at the end point of b and extends b to its opposite direction is denoted by $\overleftarrow{b}$ and is called the *backward directed extension* of b.

We now introduce three procedures that helps us to compute bridges and K.

The Procedure `coverCirclesWithLine`**:** For the following part we need a procedure that computes for a given sequence $C_1, \ldots, C_n$ of circles a line l covering as many circles as possible starting at C_1. We call that procedure `coverCirclesWithLine`. As already presented in Section 4.2 the authors of [GHMS91] have presented an approach for computing such a line l using $\mathcal{O}(n)$ time. To that end they require that $C_1, \ldots, C_n$ have the same radius or do not overlap. Later on in Section 9.3.1 we will also describe an alternative procedure using $\mathcal{O}(n^2)$ time that can help to define more precisely how the line is chosen if it is not unique. But since for the explanation of the following algorithms we only need the result of this method and are not interested how l has been computed, we use `coverCirclesWithLine` as a black-box and assume that it needs $\mathcal{O}(n)$ time. The next procedure is then based on this one and computes bridges.

The Procedure `bridge`**:** If we have a sequence $C_1, \ldots, C_n$ of circles and an index i with $1 \leq i \leq n$ we can easily compute a bridge b for a given intermediate anchor C_i. First we apply `coverCirclesWithLine` on the sequence $C_i, \ldots, C_n$ in order to obtain a line l of maximum length that covers the prefix of that sequence. Let C_k be the last circle covered by l, then we apply `coverCirclesWithLine` on the inverted sequence $C_k, \ldots, C_1$. We again get a line b. Let C_i be the circle with smallest index covered by b. Then by Definition 6.1 the line b is a bridge with anchors C_i, C_j and C_k. We denote the procedure computing the bridge for a given circle C_j by `bridge`. It results in a tuple (b, C_i, C_j, C_k). We need $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space for that procedure.

The Procedure `coverAllCircles`**:** Now we can introduce a procedure that computes a minimum sequence of bridges covering a given sequence of circles. In more detail, the procedure `coverAllCircles` returns for a given sequence of circles $C_1, \ldots, C_n$ a minimum se-

quence $b_1, \ldots, b_m$ of bridges covering the circles with respect to their order such that the intermediate anchor of a bridge b_i $(1 < i \leq m)$ corresponds to the last anchor of its predecessor b_{i-1}. To that end we follow a greedy method: Let C be the circle of the current focus that is set to C_1 at the beginning, then we compute the bridges iteratively. In the i-th step of the iteration we compute the i-th bridge b_i by applying `bridge` on $C_1, \ldots, C_n$ and C as the given intermediate anchor. Let C' be the last anchor of b_i, then we set $C := C'$ before we continue with the next iteration step. We abort the procedure when $C = C_n$ (Algorithm 1 illustrates the procedure).

Algorithm 1: coverAllCircles

input : A sequence $C_1, \ldots, C_n$ of circles.
output: A minimum sequence $b_1, \ldots, b_m$ of bridges covering $C_1, \ldots, C_n$ with respect to their order, such that the intermediate anchor of b_i $(1 < i \leq n)$ corresponds to the last anchor of its predecessor b_{i-1}.

```
1 C ← C_1;
2 m ← 1;
3 while C ≠ C_n do
4     b_i ← bridge((C_1, ..., C_n),C);
      // Let C' be the last of the last anchor of b_i.
5     C ← C';
6     m ← m + 1;
7 return b_1, ..., b_m;
```

Since a bridge (l, C_i, C_j, C_k) covers as many circles as possible from C_j to C_k which are not covered by the previous bridge, we can conclude that `coverAllCircles` returns a minimum sequence of bridges covering $C_1, \ldots, C_n$. We derive the following theorem:

Theorem 6.2. *Given a sequence $C_1, \ldots, C_n$ of circles. We can compute a minimum sequence $b_1, \ldots, b_m$ of bridges covering $C_1, \ldots, C_n$ with respect to their order in $\mathcal{O}(n)$ time.*

Proof. It remains to reason about the running time: As every circle acts at most as last anchor for one bridge and as intermediate anchor for the following bridge we can observe that each circle is covered by at most two bridges. For each bridge we need at most $\mathcal{O}(n)$ time. Further, for computing a single bridge $b = (C_i, C_j, C_j)$ we only consider the circles $C_{i-1}, \ldots, C_{k+1}$. On that account if we take a view from the side of amortization, we also can say that for each circle C_i with $1 \leq i \leq n$ we spend at most $\mathcal{O}(1)$ time per circle. Since we have n circles we derive the claim of the theorem. □

Using those bridges we can simplify a polygonal chain P to another polygonal chain P'. The next procedure describes in more detail how we do this.

The Procedure `simplifyPolygonalChain`: Given a polygonal chain $P = (p_1, \ldots, p_n)$ with corridor $\mathcal{C}$ and corresponding connecting circles $C_1, \ldots, C_n$. Further, let $b_1, \ldots, b_m$ be a minimum sequence of bridges covering P such that for two consecutive bridges b, b' with anchors (C_i, C_j, C_k) and (C_r, C_s, C_t) it is true that $i \leq j < r \leq k = s < t$. Then the procedure yields a polygonal chain P' such that P' satisfies Property 6.1. For a number j with $1 \leq j \leq n$ we define B_j to be the set containing all bridges with intermediate anchor C_j. We then intro-

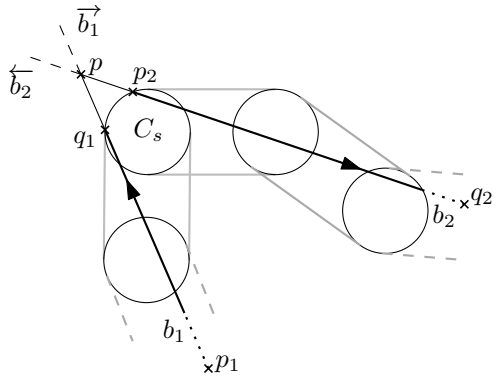


Figure 6.4: Illustration of combinable bridges. The figure shows two bridges b_1 and b_2 such that the oriented extensions intersect in a point p outside of the corridor.

duce the following definition, which helps us to describe in which cases we can connect two consecutive bridges:

Definition 6.2 (Combinable Bridges)**.** *Given a polygonal chain P of size n with corridor $\mathcal{C}$ and connecting circles $C_1, \ldots, C_n$. Let B_j and B_s be two sets of bridges such that the bridges in B_j have the anchors (C_i, C_j, C_s) and the bridges in B_s have the anchors (C_r, C_s, C_t) with $i \leq j < r \leq s < t$.*

We call two bridges $b_1 = \overline{p_1 q_1} \in B_j$ and $b_2 = \overline{p_2 q_2} \in B_s$ combinable *if $\overrightarrow{b_1}$ and $\overleftarrow{b_2}$ intersect at a point p such that $l_1 = \overline{p_1 p}$ and $l_2 = \overline{p q_2}$ do not each intersect the border of $\mathcal{C}[C_i, C_t]$ twice.*

Figure 6.4 illustrates combinable bridges and Figure 6.5 illustrates not combinable bridges. In the latter case the line $\overline{p_k p}$ intersects the right border of the corridor twice, so that b_k and b_{k+1} are not combinable.

Obviously, if $b_1, \ldots, b_m$ only consists of consecutive combinable bridges, we can use the extensions of those bridges in order to obtain a polygonal chain P' that satisfies Property 6.1. In particular the definition allows that both l_1 and l_2 intersect the border once.

Further, we make the observation for combinable bridges that if one of both line segments intersects the border only once, the other must also intersect the border so that they can intersect at the point p (see Figure 6.4). Consequently, p lies outside of $\mathcal{C}[C_i, C_t]$ and both lines intersects the same side of $\mathcal{C}[C_i, C_t]$, namely either the left or right side. On that account, if l_1 and l_2 intersect in a right kink, p must lie to the left of $\mathcal{C}[C_i, C_t]$. Analogously having a left kink at p it must lie to the right of $\mathcal{C}[C_i, C_t]$. However, since not all consecutive bridges are combinable, we apply the following procedure on the given bridges $b_1, \ldots, b_m$:

We connect two consecutive bridges $b_k = \overline{p_k q_k}$ and $b_k = \overline{p_{k+1} q_{k+1}}$ by just extending them until they intersect. In particular we use $\overrightarrow{b_k}$ and $\overleftarrow{b_{k+1}}$ in order to obtain the intersection point p. In the general case we then use p as a vertex for P', such that regarding the order of P' it lies between the vertices obtained by the bridges b_{k-1}, b_k and by the bridges b_{k+1} and b_{k+2}. But we also have to consider three special cases describing not combinable bridges. For that purpose let (C_i, C_j, C_s) be the anchors of b_k and let (C_r, C_s, C_t) be the anchors of b_{k+1}, then the special cases are:

1. b_k and b_{k+1} are parallel or
2. $\overline{p_k p}$ intersects the border of $\mathcal{C}[C_i, C_t]$ twice or

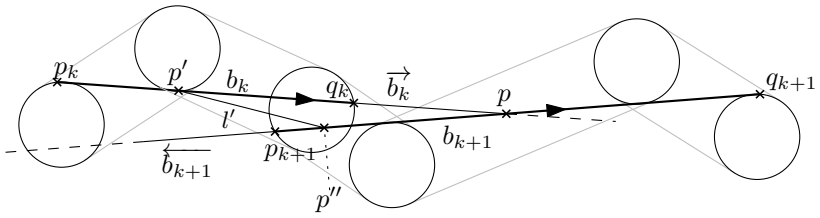


Figure 6.5: Illustration of non-combinable bridges. The figure shows two bridges b_k and b_{k+1} such that the forward directed extension of b_k intersects the corridor twice. The line segment l' shows how b_k and b_{k+1} can be connected.

3. $\overline{pq_{k+1}}$ intersects the border of $\mathcal{C}[C_i, C_t]$ twice.

In these cases we introduce a line l' that connects b_k and b_{k+1} such that l' covers at least two circles in $\mathcal{C}[C_{i-1}, C_{t+1}]$ and corresponds to the direction of the corridor (see Figure 6.5). We always can introduce l' because b_k and b_{k+1} cover at least one common circle. We then introduce the intersection points p' and p'' of l' with b_k and b_{k+1} in P' instead of p.

Obviously, we can check for the special cases in $\mathcal{O}(t-i)$ time. For each bridge b we have to do this at most twice: Once when we connect b with its predecessor and once when we connect b with its successor. Since for every circle C there are at most two bridges covering C, we can say (amortized) that for each circle C we spend $\mathcal{O}(1)$ time. Hence, we obtain a running time of $\mathcal{O}(n)$ time for connecting m bridges covering n circles.

From the definition of the algorithm it follows directly that P' satisfies Property 6.1. It remains to show the optimality of P', that is, P' has a minimum number of inflection points. We first show that the kink of two combinable bridges also fixes the kink of all others combinable bridges covering the same circles:

Lemma 6.2. *Given a polygonal chain $P = (p_1, \ldots, p_n)$ with corridor $\mathcal{C}$, and connecting circles $C_1, \ldots, C_n$. Let B_j and B_s be two sets of bridges such that the bridges in B_j have the anchors (C_i, C_j, C_s) and the bridges in B_s have the anchors (C_r, C_s, C_t) with $i \leq j < r \leq s < t$.*

If there are bridges $b_1 = \overline{p_1q_1} \in B_j$ and $b_2 = \overline{p_2q_2} \in B_s$ such that $\overrightarrow{b_1}$ and $\overleftarrow{b_2}$ intersect at a point p in a right (left) kink, then for all bridges $b_3 \in B_j$ and $b_4 \in B_s$ the following statement is true:

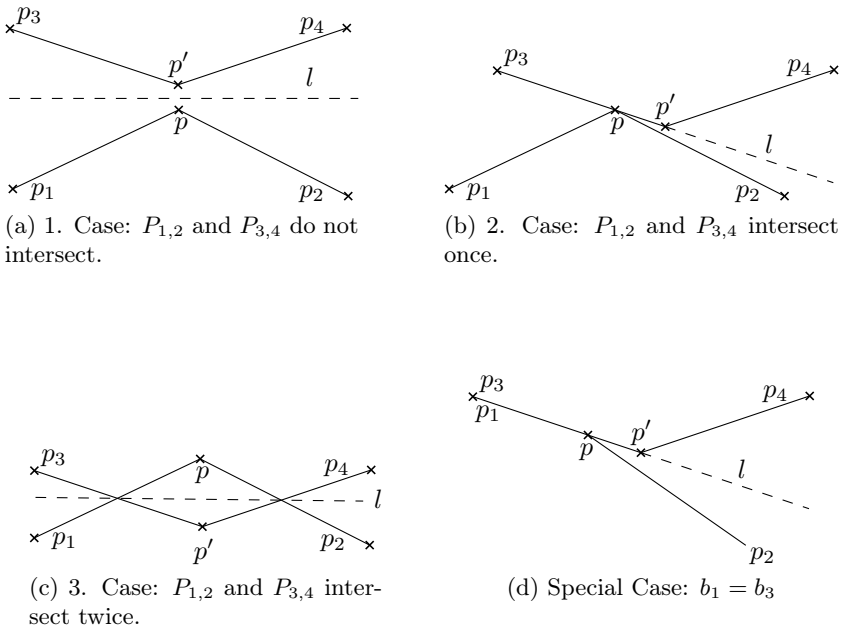
If $\overrightarrow{b_3}$ and $\overleftarrow{b_4}$ intersect, then the intersection is a right (left) kink.

Proof. Assume the contrary, that is, there are two bridges $b_1 = \overline{p_1q_1} \in B_j$ and $b_2 = \overline{p_2q_2} \in B_s$ such that $\overrightarrow{b_1}$ and $\overleftarrow{b_2}$ intersect in a right kink at a point p, but there are also two bridges $b_3 = \overline{p_3q_3} \in B_j$ and $b_4 = \overline{p_4q_4} \in B_s$ such that $\overrightarrow{b_3}$ and $\overleftarrow{b_4}$ intersect in a left kink at a point p'. We denote the polygonal chain (p_1, p, q_2) by $P_{1,2}$ and the polygonal chain (p_3, p', q_4) by $P_{3,4}$.

For the beginning assume that $b_1 \neq b_3$ and $b_2 \neq b_4$. Then there are three cases:

1. $P_{1,2}$ and $P_{3,4}$ do not intersect (see Figure 6.6a) or
2. $P_{1,2}$ and $P_{3,4}$ intersect once (see Figure 6.6b) or
3. $P_{1,2}$ and $P_{3,4}$ intersect twice (see Figure 6.6c).

In the first two cases we can easily find a line segment l, such that if $P_{1,2}$ lies on the right hand side of l, then $P_{3,4}$ lies on the left hand side of l. In the third case we define l to be the



(a) 1. Case: $P_{1,2}$ and $P_{3,4}$ do not intersect.

(b) 2. Case: $P_{1,2}$ and $P_{3,4}$ intersect once.

(c) 3. Case: $P_{1,2}$ and $P_{3,4}$ intersect twice.

(d) Special Case: $b_1 = b_3$

Figure 6.6: Illustration of the proof for Lemma 6.2.

line going through both intersection points. On that account we have found a line covering the circles $C_i, \ldots, C_t$, a contradiction to the definition of B_j and B_s.

In the case that $b_1 = b_3$, we can use $\overrightarrow{b_1}$ as l (see Figure 6.6d) and analogously if $b_2 = b_4$ we can use $\overleftarrow{b_2}$. □

Remember that a bridge b_k covers as many circles as possible starting from its intermediate anchor. Then on account of this lemma and since the intermediate anchor of a bridge b_k is the last circle covered by the predecessor b_{k-1}, we know that if $b_1, \ldots, b_m$ only consists of consecutive combinable bridges, P' is optimal. It remains to show that if there are two not combinable bridges b_k and b_{k+1}, we have to introduce a further change of left and right kinks by connecting b_k and b_{k+1} by a line segment l' as described above.

We first show that if there are two non-combinable consecutive bridges b_k and b_{k+1} then we cannot find other bridges having the same anchors such that they are combinable. To that end we first show that for a set B_j of bridges the oriented extensions of those bridges cannot intersect different sides of the corridor:

Lemma 6.3. *Given a polygonal chain P of size n with corridor $\mathcal{C}$ and connecting circles $C_1, \ldots, C_n$. Let B_j be the set of bridges with anchors (C_i, C_j, C_k).*

If $k < n$, then either for all bridges $b \in B_j$ it is true that $\overrightarrow{b}$ intersects the right side of the corridor of (C_k, C_{k+1}) or for all bridges $b \in B_j$ it is true that $\overrightarrow{b}$ intersects the left corridor of (C_k, C_{k+1}).

Proof. For each bridge $b \in B_j$ the line $\overrightarrow{b}$ must intersect either the left or the right border of (C_k, C_{k+1}), because otherwise C_k would not be the last circle that can be covered by b (see Figure 6.7). Assume that there are two bridges b_1 and b_2 in B_j, such that $\overrightarrow{b_1}$ intersect the right border of (C_k, C_{k+1}) and $\overrightarrow{b_2}$ the left border. Then the lines l_1 and l_2 extending $\overrightarrow{b_1}$ and $\overrightarrow{b_2}$ must intersect at point p forming two wedges: We can draw a line l through p and C_{k+1} such

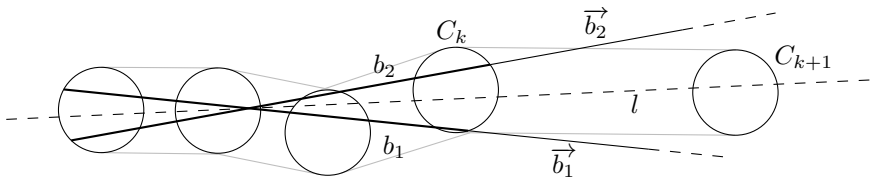


Figure 6.7: Illustration of the proof for Lemma 6.3. The forward directed extensions of the bridges b_1 and b_2 intersect the right and left border of the corridor segment (C_k, C_{k+1}), respectively.

that going along the corridor it first lies to the right of $\overrightarrow{b_1}$ and to the left of $\overrightarrow{b_2}$ and after p to the left of $\overrightarrow{b_1}$ and to the right of $\overrightarrow{b_2}$. As l lies within the wedges formed by $\overrightarrow{b_1}$ and $\overrightarrow{b_2}$, it must cover at least all circles also covered by $\overrightarrow{b_1}$ and $\overrightarrow{b_2}$. For the same reason it also must be contained in $\mathcal{C}[C_i, C_k]$. As l covers C_k and C_{k+1}, it also must be contained within $\mathcal{C}[C_k, C_{k+1}]$. Consequently, l forms a bridge covering the circles $C_1, \ldots, C_{k+1}$, which is a contradiction to the definition of B_j. □

We can introduce the same lemma for the case that we consider the backwards directed extensions of bridges. Due to symmetries, the proof can be done analogously.

Lemma 6.4. *Given a polygonal chain P of size n with corridor $\mathcal{C}$ and connecting circles $C_1, \ldots, C_n$. Let B_j be the set of bridges with anchors (C_i, C_j, C_k).*

If $1 < i$, then either for all bridges $b \in B_j$ it is true that $\overleftarrow{b}$ intersects the right corridor of (C_{i-1}, C_i) or for all bridges $b \in B_j$ it is true that $\overleftarrow{b}$ intersects the left corridor of (C_{i-1}, C_i).

Now we analyze the different cases of being not combinable. We begin with bridges that are parallel:

Lemma 6.5. *Given a polygonal chain P of size n with corridor $\mathcal{C}$ and connecting circles $C_1, \ldots, C_n$. Let B_j and B_s be two sets of bridges such that the bridges in B_j have the anchors (C_i, C_j, C_s) and the bridges in B_s have the anchors (C_r, C_s, C_t) with $i \leq j < r \leq s < t$.*

If there are bridges $b_1 \in B_j$ and $b_2 \in B_s$ such that b_1 and b_2 are parallel then there are no two bridges $b_3 \in B_j$ and $b_4 \in B_s$ such that they are combinable.

Proof. By definition of B_j and B_s the circle C_s is the last circle covered by b_1 and C_r is the first circle covered by b_2. Assume for the beginning that b_2 lies to the right of b_1 (see Figure 6.8). Then $\overrightarrow{b_1}$ must intersect the left border of the corridor segment (C_s, C_{s+1}) and $\overleftarrow{b_2}$ must intersect the right border of the corridor segment (C_{r-1}, C_r). Then from Lemma 6.4 and Lemma 6.3 we know that for all bridges $b \in B_j$ the semi-extension $\overrightarrow{b}$ must intersect the left border of (C_s, C_{s+1}) and that for all bridges $b \in B_s$ the semi-extension $\overleftarrow{b}$ must intersect the right border of (C_{r-1}, C_r).

Now assume the contrary of the claim, that is, there are two bridges $b_3 \in B_j$ and $b_4 \in B_s$ which are combinable. Then from the previous reasoning we can conclude that $\overrightarrow{b_3}$ must intersect the left border of (C_s, C_{s+1}) and $\overleftarrow{b_4}$ must intersect the right border of (C_{r-1}, C_r). On that account they cannot intersect outside of $\mathcal{C}[C_i, C_t]$, because otherwise they would not be combinable. But after $\overrightarrow{b_3}$ has intersected $\overleftarrow{b_4}$ it begins to lie between $\overleftarrow{b_4}$ and $\overleftarrow{b_2}$. Either it then intersects the right border of (C_s, C_{s+1}) in order to circumvent C_{s+1} or it covers C_{s+1}. Both are contradictions.

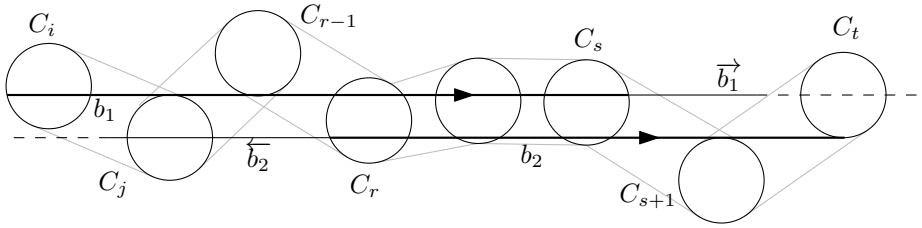


Figure 6.8: Illustration of the proof for Lemma 6.5.

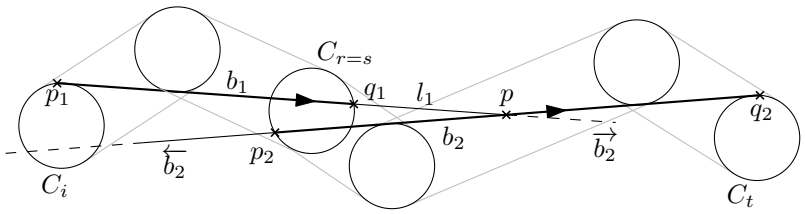


Figure 6.9: Illustration of the proof for Lemma 6.6.

We can argue analogously, if b_2 lies to the left of b_1. □

Now we consider the cases that the lines $\overline{p_k p}$ and $\overline{pq_{k+1}}$ intersect the border twice. Due to symmetry we only consider one case, and derive the lemma for the other case analogously.

Lemma 6.6. *Given a polygonal chain P of size n with corridor $\mathcal{C}$ and connecting circles $C_1, \ldots, C_n$. Let B_j and B_s be two sets of bridges such that the bridges in B_j have the anchors (C_i, C_j, C_s) and the bridges in B_s have the anchors (C_r, C_s, C_t) with $i \leq j < r \leq s < t$.*

If there are bridges $b_1 = \overline{p_1 q_1} \in B_j$ and $b_2 = \overline{p_2 q_2} \in B_s$ such that $\overrightarrow{b_1}$ and $\overleftarrow{b_2}$ intersect at a point p and $l_1 = \overline{p_1 p}$ intersects the border of $\mathcal{C}[C_i, C_t]$ twice, then there are no two bridges $b_3 \in B_j$ and $b_4 \in B_s$ such that they are combinable.

Proof. In this proof we denote the line segment $\overline{pq_2}$ by l_2. We distinguish the two cases that l_1 leaves the corridor the first time intersecting the left border or the right border. We begin with the case that l_1 intersects the left border:

It is easy to see that if l_1 leaves the corridor intersecting the left border then it must enter the corridor again intersecting the left border (see Figure 6.9). In order that l_1 can leave and enter the corridor it must intersect $\overleftarrow{b_2}$ at p after it has entered the corridor. On that account it hits $\overleftarrow{b_2}$ from the left hand side forming a left kink. Thus, in order that $\overleftarrow{b_2}$ cannot cover C_{r-1} it must intersect the right border of (C_{r-1}, C_r). Then, from Lemma 6.4 and Lemma 6.3 we know that for all bridges $b \in B_j$ the semi-extension $\overrightarrow{b}$ must intersect the left border of (C_s, C_{s+1}) and that for all bridges $b \in B_s$ the semi-extension $\overleftarrow{b}$ must intersect the right border of (C_{r-1}, C_r).

Now assume the contrary of the claim, that is, there are two bridges $b_3 \in B_j$ and $b_4 \in B - s$ which are combinable. Since both are combinable, $\overrightarrow{b_3}$ and $\overrightarrow{b_4}$ must intersect in a point p, that either lies in or to the right of $\mathcal{C}[C_i, C_t]$. Note that p cannot lie to the left of $\mathcal{C}[C_i, C_t]$ because then p would be a right kink which is a contradiction to Lemma 6.2 and the fact that $\overrightarrow{b_1}$ and $\overleftarrow{b_2}$ intersect in a left kink. But if p lies in or to the right of $\mathcal{C}[C_i, C_t]$ being a left kink, $\overrightarrow{b_1}$ must intersect the right border of (C_s, C_{s+1}), which is a contradiction to the reasoning above. □

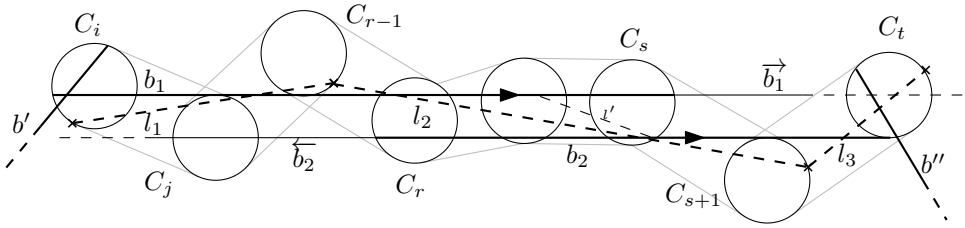


Figure 6.10: Illustration for alternative polygonal chains. Shows that the polygonal chain (l_1, l_2, l_3) has the same number pf inflection points as the solution with bridges.

The analogous lemma then says:

Lemma 6.7. *Given a polygonal chain P of size n with corridor $\mathcal{C}$ and connecting circles $C_1, \ldots, C_n$. Let B_j and B_s two sets of bridges such that the bridges in B_j have the anchors (C_i, C_j, C_s) and the bridges in B_s have the anchors (C_r, C_s, C_t) with $i \leq j < r \leq s < t$.*

If there are bridges $b_1 = \overline{p_1 q_1} \in B_j$ and $b_2 = \overline{p_2 q_2} \in B_s$ such that $\overrightarrow{b_1}$ and $\overleftarrow{b_2}$ intersect at a point p and $l_2 = \overline{pq_2}$ intersects the border of $\mathcal{C}[C_i, C_t]$ twice, then there are no two bridges $b_3 \in B_j$ and $b_4 \in B_s$ such that they are combinable.

We can combine the preceding lemmas to the following theorem:

Theorem 6.3. *Given a polygonal chain P of size n with corridor $\mathcal{C}$ and connecting circles $C_1, \ldots, C_n$. Let B_j and B_s two sets of bridges such that the bridges in B_j have the anchors (C_i, C_j, C_s) and the bridges in B_s have the anchors (C_r, C_s, C_t) with $i \leq j < r \leq s < t$.*

Two bridges $b_1 \in B_j$ and $b_2 \in B_s$ are combinable if and only if for all pairs $(b_3, b_4) \in B_j \times B_s$ it is true that b_3 is combinable with b_4.

Proof. "⇒" Follows directly from Lemma 6.5, Lemma 6.6 and Lemma 6.7.
"⇐" True due to simple logical considerations. □

Up to now we only have compared combinable bridges of the same set B_j and B_s, but it also could be that we can avoid the change between left and right kinks effected by l', if we cover the same circles as covered by B_j and B_s by three connected line segments l_1, l_2 and l_3 such that they are not of maximum length (see Figure 6.10). We use the notation of that figure. We know that the kink between b' and l_1 is always the same independent from how we choose l_1. Analogously, we can state the same for l_3 and b''. Since all pairs of $B_j \times B_s$ are not combinable (Theorem 6.3), there must be a reason why: There must be circles which can only be covered by an S-shaped polygonal chain. Independent of how we choose l_1, l_2 and l_3, we always have to introduce a change between left and right kinks. On that account it is consistent to choose l_1 as an extension of a bridge in B_j, l_3 as an extension of a bridge in B_s and l_2 as l'. Besides, the optimality of changes this also yields that P' is of minimum length under all polygonal chains satisfying Property 6.1.

Finally we can introduce the next theorem summarizing the results of this section:

Theorem 6.4. *Given a polygonal chain P of size n, then we can compute in $\mathcal{O}(n)$ time a minimum long polygonal chain P' satisfying Property 6.1 such that P' has a minimum number of changes between left and right kinks.*

Further,we can compute a smooth polyarc covering P in $\mathcal{O}(n)$, such that the polygonal chain P' on which K is based has a minimum length and a minimum number of intersection points.

Proof. The first part follows from the reasoning above.

For the second part we first obtain P' from P by applying `simplifyPolygonalChain`. Then we apply the approach of Section 6.1 on P'. We can do this because P' satisfies Property 6.1. □

7. A Generalization of the Corridor

In this chapter we take another view on the corridor C of a polygonal chain P as defined in Chapter 5. The idea is that the left side and the right side of C can be seen as two sequences of obstacles that must not be clashed by the polyarc we are looking for. More precisely we require that the elements of the right side lie to the right of the polyarc and analogously that the elements of the left side lie to the left of the polyarc. To that end we define an obstacle as:

Definition 7.1 (Obstacle)**.** *An* obstacle *is a finite open Jordan curve in the plane.*

Although we define obstacles very general as curves and do the corresponding proofs based on that assumption we normally imagine an obstacle as a line segment or a circular arc. Especially on the implementation level it is crucial to have simple geometrical primitives describing obstacles in order to keep the obstacles usable.

Since later on an arc A goes through a field of obstacles, such that a pre-defined set of those obstacles lies to the left and the others to the right of A we distinguish a priori between *left obstacles* and *right obstacles*. We define whether an obstacle lies on the correct side of an arc:

Definition 7.2. *Let A be an arbitrary arc with corresponding circle C, then we say that a right obstacle r* lies on the correct side of *A if the following three statements are true:*

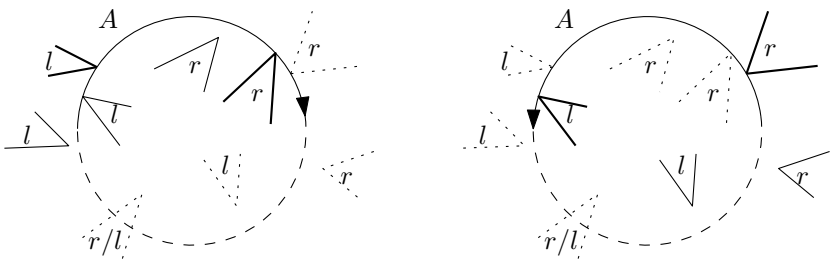
1. *If A is clockwise oriented, then r does not lie outside of C.*
2. *If A is counterclockwise oriented, then r does not lie inside of C.*
3. *If A is a straight line segment, then r does not lie to the left of C.*

Analogously, we say that a left obstacle l lies on the correct side of *A if the following three statements are true:*

1. *If A is clockwise oriented, then l does not lie inside of C.*
2. *If A is counterclockwise oriented, then l does not lie outside of C.*
3. *If A is a line segment, then l does not lie to the right of C.*

The definition is illustrated in Figure 7.1. We have to introduce the third requirement because a line segment has both a clockwise orientation and a counterclockwise orientation corresponding circle. On that account the definition is well-formed. Further, due to the definition we also allow obstacles to touch arcs, that is, an arc and an obstacle may have points in common. More precisely we define:

Definition 7.3. *An obstacle o* touches *an arc A if o lies on the correct side of A and A and o have at least one point in common.*



(a) A is clockwise oriented. (b) A is counterclockwise oriented.

Figure 7.1: Illustration of arcs and obstacles. Wedges marked with r are right obstacles and wedges marked with l are left obstacles. The dotted obstacles do not lie on the correct side of A, while the others do. Further bold obstacle touch A.

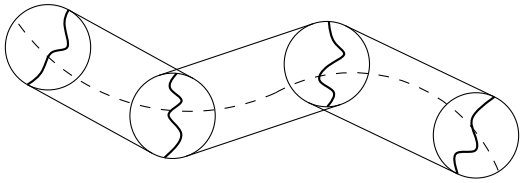


Figure 7.2: Illustration of the idea of gates.

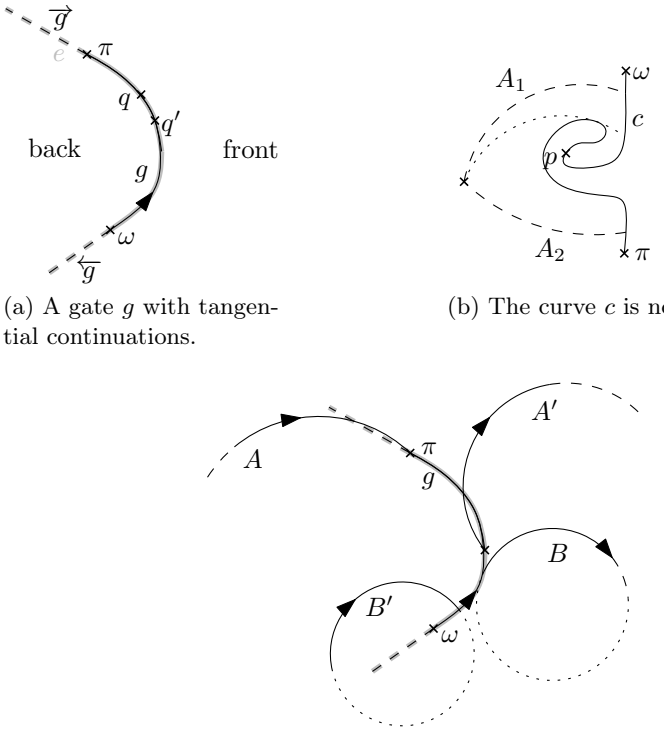
Due to this definition (see Figure 7.1), we only call the relation that A and o have a common point a *touching relation* if the obstacle also lies on the correct side of A. In all cases that do not satisfy the requirements of the definition, we say that o *clashes* with A. This also includes the case that o does not lie on the correct side of A and does not have any points with A in common. Later on we define the clashing of an obstacle and an arc more precisely in order to have terminology for defining the position of a clash.

But first we want to introduce the next concept: We know that the polyarc we are looking for must pass the connecting circles of the corridor segments. In particular that means that for each connecting circle C we can find a curve c contained in C which the chain must pass through (see Figure 7.2). In accordance with [DRS08] we call those curves *gates*:

Definition 7.4. *We call an oriented finite curve $g = (\pi, \omega)$ with starting point π and end point ω a* gate *if*

- *the tangential continuations $\overrightarrow{g}$ and $\overleftarrow{g}$ do not intersect each other, and*
- *$\overleftarrow{g}$ and $\overrightarrow{g}$ do not intersect g, and*
- *for all points p in the plane and all points p_1 and p_2 on g it is true, that for all arcs $A_1 = (p, \cdot, p_1)$ and $A_2 = (p, \cdot, p_2)$ that do not intersect g there is for any point p' on g between p_1 and p_2 an arc $A' = (p, \cdot, p')$ that lies between A_1 and A_2 and does not intersect g.*

We call the curve assembled by $\overleftarrow{g}$, g and $\overrightarrow{g}$ having the same orientation as g the extension *of g. Since g is oriented, we say that a point $q \in g$ lies to the* left *of a point $q' \in g$ if going from π to ω we first visit q' and then q.*



(a) A gate g with tangential continuations.

(b) The curve c is not a gate.

(c) A ends wrongly at g and A' starts wrongly at g. The arc B touches g and the arc B' crosses g.

Figure 7.3: Illustration of a single gate. The gray curve illustrates the extension of the gate.

The definition is illustrated in Figure 7.3a. The first two requirements ensure that a gate g does not form any kinds of loops, while the third requirement ensures that the reachable segment on g regarding a point p can be described by a closed interval. Note that the third requirement also demands that the arcs do not intersect g (see Chapter 2, curves). For example Figure 7.3b shows a curve that cannot be a gate: We cannot find an arc between A_1 and A_2 which reaches p without intersecting c. Later on we will show for several geometric primitives that they fit the definition of gates. But first we want to talk about some terminology describing gates:

We call the right side of g the *front* of g and the left side of g the *back* of g. Later on we use the front of a gate g as the side of g at which an arc may start and analogously the back of g as the side where an arc may end. We say that an arc A starts or ends *wrongly* at g if it intersects the extension of g after/before it starts/ends at g (see arcs A and A' in Figure 7.3c). For this definition keep in mind that the extension of g also consists of g itself.

If an arc A does not start or end wrongly at a gate g, we distinguish two types of connections between A and g: We say that an arc A *touches* a gate g if A starts or ends at g and its corresponding circle lies only on one side of the extension of the gate (see arc B in Figure 7.3c). We also write $g \parallel A$ in order to express that g touches A. If the corresponding circle lies on both sides of the gate (see arc B' in Figure 7.3c) then we say that A crosses g. We also write $g \not\parallel A$ in order to express that g crosses A.

The concept of gates is not new, but has been already used in [DRS08] by assuming that a gate is a line segment. Due to the simplicity of line segments, we also first use line segments

as gates and assume that they are pre-defined in our corridor. In Section 9.2 we also consider gates based on circular arcs, in order to circumvent the problem of choosing good gates which arises in [DRS08]. We now show that line segments are gates by proving that a subset of curves are gates.

Lemma 7.1. *Let $g = (\pi, \omega)$ be a finite non-self-intersecting curve without inflection points such that the tangential continuations of g do not intersect each other and do not intersect g, then g is a gate.*

Proof. By definition of g the tangential continuations of g do not either intersect each other or g. It remains to show the third requirement of a gate. For that purpose consider an arbitrary point p in the plane and two distinct points p_1 and p_2 on g. Further, let $A_1 = (p, \cdot, p_1)$ and $A_2 = (p, \cdot, p_2)$ be two arbitrary arcs that do not intersect g (see Figure 7.4).

Now assume that there is a point p' between p_1 and p_2 on g that is not reachable by an arc A' which lies between A_1 and A_2 such that it does not intersect g.

First assume that A_1 and A_2 intersect at another point t. We draw an arc A' from p through t to p' (see Figure 7.4a). Since arcs can only intersect twice, the only intersection points of A' with A_1 and A_2 are p and t. On that account and due to p' lies to the left of p_1 and to the right of p_2, we can conclude that from p to t the arc A_1 lies firstly to the left of A' and afterwards to the right of A'. Analogously, from p to t the arc A_2 lies to the right of A' and then to the left of A'. Consequently, A' lies between A_1 and A_2. Due to the loss of inflection points of g we also know that A' does not intersect g. Note that the arguments are independent from the orientation of the arcs.

If A_1 and A_2 only intersect in p, we consider the corresponding circle C_1 and C_2 of A_1 and A_2 and distinguish two cases:

If C_1 and C_2 intersect in another point t (see Figure 7.4b), we draw an arc A' from p through p', such that it corresponding circle C' also goes through t. Since C' intersects A_1 and A_2 twice, A' can only intersect A_1 and A_2 in p. As p' lies between p_1 and p_2, A' must lie to the left of A_1 and to the right of A_2. Consequently, A' lies between A_1 and A_2 and because of g does not have any inflection points the arc A' cannot intersect g. Again, it was not necessary to consider the orientation of the arcs.

If C_1 and C_2 only intersect in p, we can find a circle C' that goes through p and p' such that C' only intersects C_1 and C_2 once (see Figure 7.4c). Then either C_1 or C_2 is fully contained in C', while the other fully contains C'. On that account we can use C' in order to obtain an arc A' that lies between A_1 and A_2. Again, we apply the argument of inflection points in order to guarantee that A' does not intersect g. □

We can directly conclude that

Corollary 7.1. *An oriented line segment $\overline{\pi\omega}$ is a gate g with starting point π and end point ω.*

We do not want to generalize the corridor too much. For example we still want that two gates representing two connecting circles of the same corridor segment can be connected by a line segment without clashing obstacles or that obstacles between two consecutive gates do not overlap each other. We formalize this and other technical requirements as follows:

Definition 7.5 (Feasible Constellation)**.** *Given two gates $g_1 = (\pi_1, \omega_1)$ and $g_2 = (\pi_2, \omega_2)$ with extensions e_1 and e_2, a sequence $L = (l_1, ..., l_m)$ of left obstacles and a sequence $R = (r_1, \ldots, r_m)$ of right obstacles. We call the tuple (g_1, g_2, L, R) a* feasible constellation of two consecutive gates with obstacles *if*

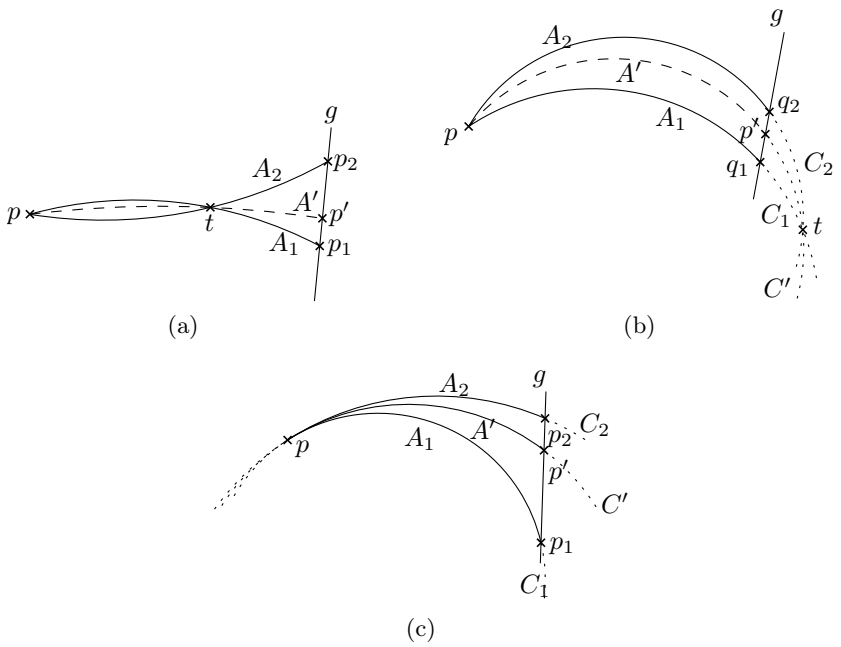


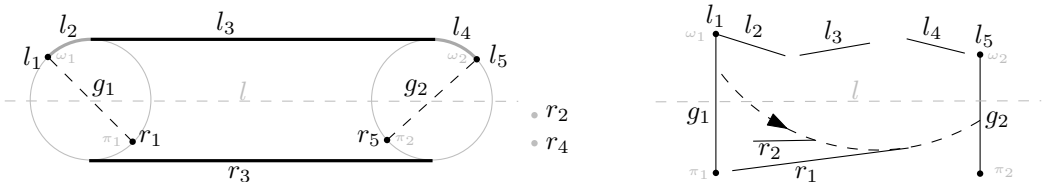
Figure 7.4: Illustration of the proof for Lemma 7.1. The arc A' is dashed.

1. $r_1 = \pi_1$, $r_m = \pi_2$, $l_1 = \omega_1$, $l_m = \omega_2$.
2. g_1 *and* g_2 *do not have any common points.*
3. *There is an oriented line* l *crossing first* g_1 *and then* g_2 *such that*
   a) l *first hits the back of* g_1 *and then the back of* g_2 *and*
   b) l *has only one common point with each gate and*
   c) *there is no left obstacle lying to the right of* l *and*
   d) *there is no right obstacle lying to the left of* l.
4. *There is no arc touching* e_1, l_j, l_i *and then* e_2 *for any* $i < j$.
5. *There is no arc touching* e_1, r_j, r_i *and then* e_2 *for any* $i < j$.
6. *Left and right obstacles do not have common points.*

We often say more shortly a feasible constellation (g_1, g_2, L, R) with extensions e_1 and e_2.

Further, let (g_1, g_2, L, R) *and* (g_2, g_3, L', R') *be two feasible constellations with* $g_1 \neq g_3$, $L \cap L' = \emptyset$ *and* $R \cap R' = \emptyset$ *then we call the composition* $(g_1, g_3, L + L', R + R')$ *also* a feasible constellation.

Figure 7.5 illustrates the definition: In particular the third requirement also states that obstacles can have common points with l. Requirement four and five say that there is a particular order arcs can touch left respective right obstacle. It also ensures that obstacles between two consecutive gates cannot overlap. Figure 7.5b shows two gates with obstacles where the fourth requirement is satisfied but the fifth is not satisfied, because we can find an arc connecting g_1 with g_2 which touches first r_2 and then r_1.



(a) A feasible constellation: r_2 and r_4 are dummy obstacles.

(b) Not a feasible constellation.

Figure 7.5: Illustration of a feasible constellation.

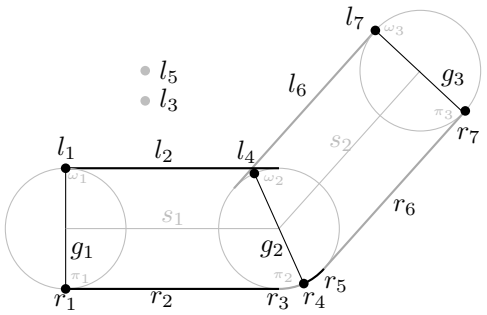


Figure 7.6: Illustration of two corridor segments translated into gates and obstacles. The obstacles are depicted as black and gray bold segments and points. The points l_3 and l_5 are dummy obstacles. Further $l_1 = \omega_1$, $l_4 = \omega_2$, $l_7 = \omega_3$, $r_1 = \omega_1$, $r_4 = \omega_2$ and $r_7 = \omega_3$.

Further, we require that there is always the same number of left obstacles and right obstacles (which is not satisfied in Figure 7.5b). Later we use this requirement in order to define valid arcs. Note that this is not a real restriction, because we always can add a constant number of left and right obstacles that never clash with arcs in order to satisfy the requirement. We call such obstacles also *dummy-obstacles*. Figure 7.5a illustrates how a corridor segment can be translated into a feasible constellation where we have chosen the gates arbitrarily.

Figure 7.6 shows the translation of two consecutive corridor segments into gates and obstacles: In particular the figure shows the three feasible constellations

$$c_1 = (g_1, g_2, (l_1, \ldots, l_4), (r_1, .., r_4)),\ c_2 = (g_2, g_3, (l_4, \ldots, l_7), (r_4, \ldots, r_7))$$
$$\text{and } c_3 = (g_1, g_3, (l_1, \ldots, l_7), (r_1, \ldots, r_7)).$$

The last constellation is feasible by definition because it is composed by the feasible constellations c_2 and c_3. Note that a composed feasible constellation can consist of obstacles that may overlap.

Since in Section 9 we discuss in more detail how one can choose gates appropriately, the gates are depicted arbitrarily in the figures. Further, the presented translation of two consecutive corridor segments induces the translation of a whole corridor $\mathcal{C}$ into feasible constellations. It follows directly from the translation that the resulting feasible constellations for the single corridor segments can be composed to one feasible constellation that comprises the whole corridor $\mathcal{C}$.

Analyzing feasible constellation we often get in touch with arcs that clash obstacles. We already have defined the notion of clashing but we also want to have the possibility to talk about the location of the clash:

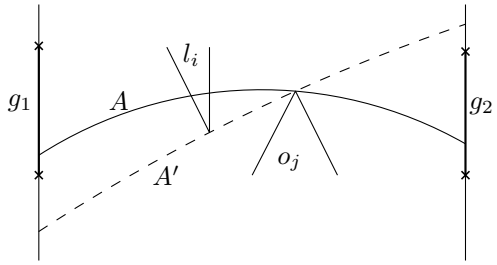


Figure 7.7: Clashing location of an arc. The arc A' is witness that the left obstacle l_i clashes the arc A before o_j touches A.

Definition 7.6 (Clashing Location)**.**
Given a feasible constellation $(g_1, g_2, L = (l_1, ..., l_m), R = (r_1, \ldots, r_m))$ with extensions e_1 and e_2. Let A be an arbitrary arc and let o_j be an arbitrary right or left obstacle in $L \cup R$ touching A. Further, let l_i be a left obstacle clashing A. Then we say

1. *the left obstacle l_i clashes A* before *o_j if there is an arc A' that first touches e_1, then l_i, then o_j and finally e_2.*
2. *the left obstacle l_i clashes A* after *o_j if there is an arc A' that first touches e_1, then o_j, then l_i and finally e_2.*

We define analogously the same terminology for right obstacles.

Figure 7.7 illustrates the definition. Note that an obstacle o_i can only clash before or after another obstacle o_j regarding an arc A if o_j touches A. Further, by definition it is possible that o_i clashes A before and after o_j. Based on this terminology we can introduce *valid arcs*:

Definition 7.7. *Given two gates g_1 and g_2 with extensions e_1 and e_2, left obstacles $L = (l_1, ..., l_m)$ and right obstacles $R = (r_1, ..., r_m)$. Then an arc $A = (p_1, p_2, p_3)$ is called* valid *with respect to (g_1, g_2, L, R) if all of the following requirements are met:*

(1) p_1 lies on g_1 and p_3 lies on g_2.

(2) A does not start at g_1 wrongly and A does not end at g_2 wrongly.

(3) All obstacles of L lie to the left of A.

(4) All obstacles of R lie to the right of A.

(5) If two obstacles $o_i, o_j \in L \cup R$ with $i < j$ touch A, then the touching point of o_i lies before the touching point of o_j on A.

(6) If two obstacles $r_i \in R$ and $l_i \in L$ (of the same index) touch A, then either r_i touches A before l_i does or l_i touches A before r_i does.

Requirement (2) says that an arc A must respect the forward direction of the gate it reaches. In particular by requirements (5) and (6) of the definition we also want that there is certain kind of equal valued pairs consisting of one left obstacle and one right obstacle having the same index, that is, the definition does not require a certain touching order between a left

and right obstacle having the same index. This is more a technical detail, which is later on used in the presented algorithms.

If an arc A passes through a sequence of gates, we denote the sub-arc of A connecting two gates g_i, g_j of this sequence by $A[g_i, g_j]$. Then we can formalize the problem we want to solve as follows:

Problem 7.1.
Given n feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), ..., (g_{n-1}, g_n, L_n, R_n) where $L_i = (l_{i,1}, \ldots, l_{i,k_i})$ and $R_i = (r_{i,1}, \ldots, r_{i,k_i})$ with $k_i \in \mathbb{N}$.

Then we want to find a sequence $A_1 = (p_1, \cdot, q_1), \ldots, A_m = (p_m, \cdot, q_m)$ of arcs, such that m is minimized and

1. *all arcs start and end at gates, and*
2. *p_1 lies on g_0 and q_m lies on g_n, and*
3. *$p_{i+1} = q_i$ for all $i \in \{1, \ldots, m-1\}$, and*
4. *for two consecutive gates g_{i-1} and g_i connected by the arc A, the sub-arc $A[g_{i-1}, g_i]$ is valid with respect to (g_{i-1}, g_i, L_i, R_i), and*
5. *each arc A_i $(1 \leq i \leq m)$ crosses all gates lying between the starting gate and the end gate of A_i.*

The fourth requirement states that between two consecutive gates g_{i-1} and and g_i an arc can only clash obstacles that belong to the feasible constellation (g_{i-1}, g_i, L_i, R_i). On that account arcs can clash with obstacles only locally: By means of that requirement we achieve that two overlapping corridor segments do not influence each other, but the obstacles clash only parts of the polyarc that actually passes the corridor segment.

So far we have formalized the problem of covering a polygonal chain by a polyarc using the terminology of gates and obstacles, whereat we make the restriction that arcs are only allowed to start and end at predefined gates. Later, we loosen this restriction, but in the next chapter we first want to explain how one can solve Problem 7.1.

8. Basic Algorithms for the Generalized Corridor

After we have described in Chapter 7 a way to generalize the corridor $\mathcal{C}$, which we have introduced in Chapter 5, we now present basic algorithms that work on that generalization. In particular in this chapter we assume that we are given a sequence of feasible constellations $(g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), \ldots, (g_{n-1}, g_n, L_n, R_n)$ based on $n+1$ consecutive gates. Each constellation represents one corridor segment of $\mathcal{C}$. In Section 8.1 we first introduce a fundamental algorithm that helps us to compute circular arcs connecting g_0 with its successors $g_1, \ldots, g_n$ such that those resulting arcs respect the order of $\mathcal{C}$. Then based on that algorithm we explain in Section 8.2 how one can gain a polyarc (not necessarily smoothly connected) covering $\mathcal{C}$.

In order to express how an arc is clashed by obstacles we introduce the following terminology describing sequences of touching events by regular expressions: Given two gates $g_1 = (\pi_1, \omega_1)$ and $g_2 = (\pi_2, \omega_2)$, then we define the sequence $\sigma(A)$ of touching events of A over the alphabet $\Sigma = \{r, l, \pi_1, \pi_2, \omega_1, \omega_2\}$, where the letters have the following meaning:

r: A is touched by a right obstacle.

l: A is touched by a left obstacle.

π_1: A ends at the right end point of g_1 (Analogously π_2, ω_1 and ω_2).

In order to describe the sequence $\sigma(A)$ of an arc $A = (p, \cdot, q)$ we use regular expressions. For example $\sigma(A) = [r]l^*r\pi_2$ means that going from p to q along A, A optionally first touches one right obstacle, then an arbitrarily long sequence of left obstacles (empty sequence included), afterwards one right obstacle and finally A ends at π_2. We also describe sequences by terms having the shape $..x..y..z \in \sigma(A)$, where x, y and z are placeholders for single letter of Σ. The expression can be interpreted as: We can remove letters from the sequence $\sigma(A)$ such that the sequence xyz remains. If .. is placed at the beginning or at the end of a sequence, this means that the sequence does not start directly at the starting gate or at the end gate. For example $..l..r \in \sigma(A)$ means, that A is touched by a left obstacle, such that the touching point does not lie on the starting gate, and then A is touched by an obstacle from the right side, which also can happen on the end gate. Note that for a reasonable choice of obstacles that the only obstacles lying on a gate should be the starting and end point of the same gate.

8.1 One Starting Gate and Several End Gates

For solving Problem 7.1 we analyze a similar problem that considers only one starting gate g_0 and several end gates $g_1, \ldots, g_n$ in order to compute the last reachable gate from g_0. To that end we try to compute to for each gate g_i with $1 \leq i \leq n$ a valid arc that goes from g_0 to g_i:

Problem 8.1. *Given n feasible constellations $(g_0, g_1, L_1, R_1), \ldots, (g_{n-1}, g_n, L_n, R_n)$ where $L_i = (l_{i,1}, \ldots, l_{i,k_i})$ and $R_i = (r_{i,1}, \ldots, r_{i,k_i})$ with $k_i \in \mathbb{N}$.*

Then we want to find a sequence $A_1 = (p_1, \cdot, q_1), \ldots, A_m = (p_m, \cdot, q_m)$ of arcs, such that m is maximized and

1. *p_i lies on g_0 for all $i \in \{1, \ldots, n\}$, and*
2. *q_i lies on g_i, and*
3. *each arc A_i $(1 \leq i \leq m)$ crosses all gates lying between g_0 and g_i, and*
4. *for two consecutive gates g_{i-1} and g_i connected by the arc A the sub-arc $A[g_{i-1}, g_i]$ is valid with respect to (g_{i-1}, g_i, L_i, R_i).*

At the end of this section we present an algorithm for solving this problem using $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ storage. To that end we analyze the problem step by step: First we consider only two gates g_1 and g_2 and give criteria how they can be connected by valid arcs. Afterwards we generalize this knowledge in order to solve Problem 8.1.

8.1.1 Two Consecutive Gates

In this part we consider one feasible constellation (g_1, g_2, L, R) and describe how g_1 and g_2 can be connected by valid arcs. If there is a valid arc connecting g_1 and g_2 we have to answer which of the points on g_2 are reachable from g_1. To that end we first define sets containing all reachable points:

Definition 8.1 (Reachable Points)**.** *Given a feasible constellation (g_1, g_2, L, R). For a point p on g_1 the set $\overrightarrow{\mathcal{R}}(p)$ contains all points q on g_2 for which a valid arc $A = (p, \cdot, q)$ exists. Analogously, we define for a point q on g_2 the set $\overleftarrow{\mathcal{R}}(q)$, which contains all points p on g_1 for which a valid arc $A = (p, \cdot, q)$ exists.*

We call these sets reachable points *of p respective of q.*

Further, we define the reachable points of g_1 as $\overrightarrow{\mathcal{R}} = \bigcup_{p \in g_1} \overrightarrow{\mathcal{R}}(p)$ and the reachable points of g_2 as $\overleftarrow{\mathcal{R}} = \bigcup_{q \in g_2} \overleftarrow{\mathcal{R}}(q)$.

Using the following lemma we show that the reachable points of a point p on g_1 or on g_2 describe a closed line segment. Consequently, we can describe the reachable points of p by a closed interval.

Lemma 8.1. *Given a feasible constellation (g_1, g_2, L, R). For each point p on g_1 its set of reachable points $\overrightarrow{\mathcal{R}}(p)$ is either empty or it can be described by an interval $[r, l] \subseteq [0, 1]$ on g_2 such that $\forall \lambda \in \mathbb{R} : \lambda \in [r, l] \Leftrightarrow g_2(\lambda) \in \overrightarrow{\mathcal{R}}(p)$.*

Proof. Obviously, $\overrightarrow{\mathcal{R}}(p)$ can be empty for a point p on g_1. If $\overrightarrow{\mathcal{R}}(p)$ is not empty, the claim follows directly from the definition of a gate. Since obstacles are allowed to touch obstacles the interval is closed. □

For the opposite direction we can prove the very same:

Lemma 8.2. *Given a feasible constellation (g_1, g_2, L, R). For each point q on g_2 its set of reachable points $\overleftarrow{\mathcal{R}}(q)$ is either empty or can be described by an interval $[r, l] \subseteq [0, 1]$ on g_1 such that $\forall \lambda \in \mathbb{R} : \lambda \in [r, l] \Leftrightarrow g_1(\lambda) \in \overleftarrow{\mathcal{R}}(q)$.*

Proof. Can be proven analogously to Lemma 8.1. □

From both lemmas we can derive the following corollary:

Corollary 8.1. *Both the reachable points $\overrightarrow{\mathcal{R}}$ of g_1 and the reachable points $\overleftarrow{\mathcal{R}}$ of g_2 can be described by closed intervals $[r_1, l_1]$ and $[r_2, l_2]$.*

The next step is, that we explain how one can compute the reachable points, that is, we show how one can find unique arcs reaching the extremes of the reachable points. Due to symmetry we restrict the following analysis on the leftmost point that is reachable from g_1.

Since there can be several arcs having the same end points, we introduce the concept of *tight* arcs in order to obtain unique arcs. One can imagine a tight arc as a kind of elastic strap that is tighten by pulling its ends away from each other. Such a strap normally forms a line segment ($\Phi(A) = 0$), if there is not any reason why it should behave otherwise. In the case that there is a reason for bending, than a tight arc bends as little as possible. The following definition formalizes this observation:

Definition 8.2 (Tight Arc)**.** *Given a feasible constellation (g_1, g_2, L, R). Let $A = (p, \cdot, q)$ be a valid arc connecting g_1 and g_2. Then we call A a* tight arc*, if for all other valid arcs $A' = (p, \cdot, q)$ connecting the same points holds that the outgoing direction of A' is greater than A by amount: $|\Phi(A')| > |\Phi(A)|$.*

Note that the definition is consistent, because two arcs connecting the same end points and having the same outgoing direction are equal. From this observation we can directly conclude the following lemma:

Lemma 8.3. *Given a feasible constellation (g_1, g_2, L, R) and a tight arc $A = (p, \cdot, q)$ connecting g_1 with g_2, then A is unique.*

The following lemma states that a tight arc only bends if it is necessary:

Observation 8.1. *Given a feasible constellation (g_1, g_2, L, R) and a tight arc $A = (p, \cdot, q)$ connecting g_1 with g_2. Then the following statements are true:*

1. *If A is not touched by a left obstacle, then $\Phi(A) \leq 0$.*
2. *If A is not touched by a right obstacle, then $\Phi(A) \geq 0$.*
3. *If A is touched neither by a left obstacle nor by a right obstacle, then $\Phi(A) = 0$.*
4. *If $\Phi(A) < 0$ then there is at least one right obstacle that touches A between p and q $(..r.. \in \sigma(A))$*
5. *If $\Phi(A) > 0$ then there is at least one left obstacle that touches A between p and q $(..l.. \in \sigma(A))$*

Having this vocabulary we now characterize the tight arcs by sequences of obstacles touching them. Since we do not restrict the outgoing direction of an arc, there can occur one special case where the obstacles are far away from the given gates. To that end we define the left boundary respective the right boundary of two gates g_1 and g_2 as:

Definition 8.3 (Left Boundary/Right Boundary)**.**
Given a feasible constellation $(g_1 = (\pi_1, \omega_1), g_2 = (\pi_2, \omega_2), L, R)$. The left boundary of g_1 and g_2 is the arc A connecting ω_1 with ω_2 such that A touches g_1 or g_2 and its outgoing direction is maximized.

Analogously we define the right boundary of g_1 and g_2.

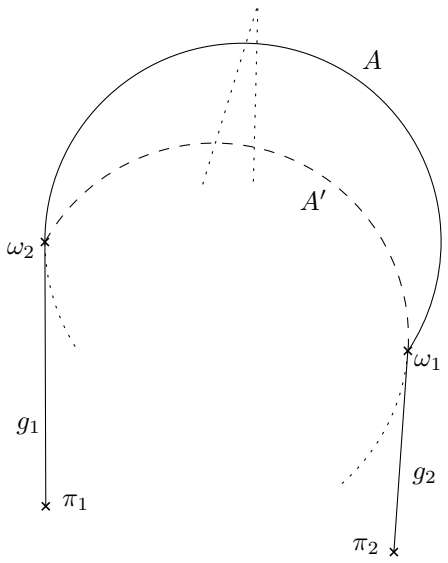


Figure 8.1: Illustration of the left boundary of two gates

Figure 8.1 illustrates the left boundary: There is always one arc A connecting ω_1 with ω_2 such that A touches g_1 and there is always one arc A' connecting ω_1 with ω_2 such that A' touches g_2. The left boundary arc is the arc with maximal outgoing direction (In Figure 8.1 arc A). Later on we will explain how one can handle the special case that there is an obstacle clashing the boundary, but for the beginning we assume that these boundaries are not clashed by any obstacles. Then we can use the next lemma to obtain for a point p on g_1 the rightmost point on g_2 that is reachable from p:

Lemma 8.4 (Fixed Starting Point)**.**
Given a feasible constellation $(g_1 = (\pi_1, \omega_1), g_2 = (\pi_2, \omega_2), L, R)$, such that g_2 is reachable from g_1 and the left boundary of g_1 and g_2 is not clashed by a right obstacle. Let p be an arbitrary point on g_1 and let q be an arbitrary point in $\overrightarrow{\mathcal{R}}(p)$.

Then q is the rightmost point reachable from p if and only if for the tight arc $A = (p, \cdot, q)$ it is true that

S1: $\Phi(A) = 0 \wedge \pi_2 = q$, or

S2: $\Phi(A) < 0 \wedge ..r..\pi_2 \in \sigma(A)$, or

S3: $\Phi(A) \leq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge ..r \in \sigma(A)$, or

S4: $l..r \in \sigma(A)$.

Proof. "⇒": Assume that q is the rightmost point reachable from p. Then a tight arc $A = (p, \cdot, q)$ exists, because otherwise q would not be reachable from p. By Lemma 8.3, this arc is unique. We make a case distinction between $q = \pi_2$ and $q \neq \pi_2$.

First let $q = \pi_2$, then A touches at least one right obstacle, namely π_2. If A touches also a left obstacle between p and q, then $S4$ is true. If A is not touched by a left obstacle, then by Observation 8.1 for the outgoing direction of A it is true that $\Phi(A) \leq 0$. If $\Phi(A) = 0$ then $S1$ is true and if $\Phi(A) < 0$ then by Observation 8.1 the statement $S2$ is true.

So let $q \neq \pi_2$, then we only have to show that A is of type $S3$ or $S4$: Since $q \neq \pi_2$, the arc A must touch another right obstacle, because otherwise q is not the rightmost reachable point. If A is touched by left obstacles before it touches a right obstacle, then $S4$ is true. So let left obstacles only touch A, after all right obstacles have touched A: $rr^*l^* = \sigma(A)$. We denote the last right obstacle touching A by r_i.

Assume A does not touch g_1 and g_2 when it is clockwise oriented, that is, $S3$ does not hold. Then we can draw an arc A' from p to a point q' on g_2, such that q' lies to the right of q. Further, we choose A' such that it first lies to the left of A' and intersects A after r_i has touched A_i (see Figure 8.2a). Note that this is only possible, because we require, that no right obstacle clashes the left boundary arc of g_1 and g_2, and if A is clockwise oriented then it does not touch g_1 or g_2. Since A' first lies to the left of A and intersects A after r_i has touched A and $rr^*l^* = \sigma(A)$, we can guarantee that A' is a valid arc. We only have to choose the distance between q and q' appropriately small. But then the existence of A' is a contradiction to that q is the rightmost point reachable from p.

"$\Leftarrow$": Assume that for $A = (p, \cdot, q)$ one of the four statements is true and A is a tight arc with respect to p and q, but q is not the rightmost point reachable from p. Obviously $S1$ and $S2$ cannot be true for A, because then q would be π_2, the rightmost reachable point from g_1. We therefore only consider $S3$ and $S4$. In both cases A touches at least one right obstacle different to π_1.

If q is not the rightmost point reachable from p, then there is another point q' satisfying this property (By Lemma 8.1 the reachable points of a given point can be described by a closed interval). Let $A' = (p, \cdot, q')$ be an arbitrary valid arc connecting p and q'.

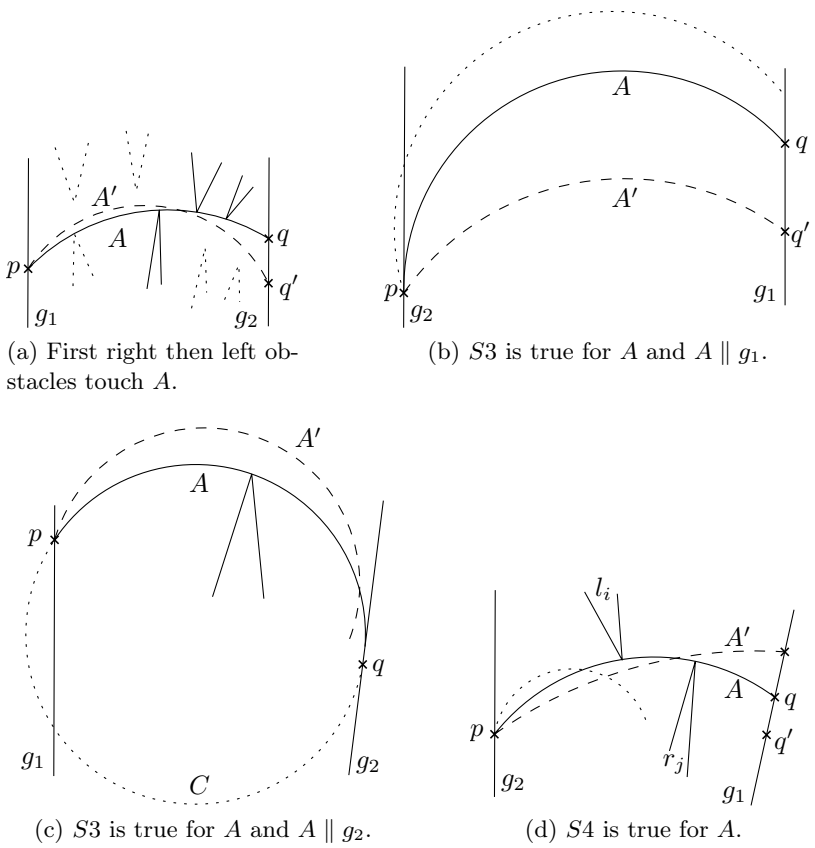
Assume that $S3$ is true and A touches g_1 (see Figure 8.2b). In order for A' to reach a point q' that lies to the right of q, A' must start with another direction at g_1 as A has. If it starts such that it lies first to the left of A, it must intersect the extension of g_1: A' has not the same direction at p as g has. Due to the differentiability of g_1 then A' must start at g_1 such that it intersects the extension of g_1 again in order to reach g_2. Consequently, A' starts wrongly from g_1.

On that account, assume that A' firstly lies to the right of A. Since A and A' have the point p in common, they can only intersect once more. But this is not possible, because then A' could not reach q' anymore. On that account A' completely lies to the right of A and the only way for A to reach a right obstacle is at p. This is a contradiction to A touches a right obstacle different to p.

Now assume that $S3$ is true and A touches g_2 (see Figure 8.2c). Then A' lies first to the left of A in order to circumvent the right obstacle touching A. After A has touched the last right obstacle, A' intersects A the second time. Since A is clockwise oriented, A' then is contained in the corresponding circle of A. Due to A only touches g_2, A' cannot reach g_2.

Finally let $S4$ be true for A (see Figure 8.2d), that is, A first touches a left obstacle l_i and then a right obstacle r_j. Consider the case that A' lies first on the left hand side of A. Consequently it must intersect A before A touches l_i. However,from that moment on it remains to the right of A, so that it clashes with r_j. Thus, A' must first lie to the right side of A. In order to circumvent r_j it intersects A once. But then A' has intersected A twice in total and must therefore stay on the left hand side of A. On that account, A cannot reach a point q' to the right of q. □

The following lemma shows the symmetric version of Lemma 8.4 and can be maximum analogously.



(a) First right then left obstacles touch A.

(b) $S3$ is true for A and $A \parallel g_1$.

(c) $S3$ is true for A and $A \parallel g_2$.

(d) $S4$ is true for A.

Figure 8.2: Illustration of the proof for lemma 8.4.

Lemma 8.5. *Fixed Starting Point (Symmetric Version) Given a feasible constellation* $(g_1 = (\pi_1, \omega_1), g_2 = (\pi_2, \omega_2), L, R)$*, such that* g_2 *is reachable from* g_1 *and the left boundary of* g_1 *and* g_2 *is not clashed by a right obstacle. Let* q *be an arbitrary point on* g_2 *and let* p *be an arbitrary point in* $\overleftarrow{\mathcal{R}}(q)$*.*

Then p *is the rightmost point reachable from* q *if and only if for the tight arc* $A = (p, \cdot, q)$ *holds that*

$S1'$: $\Phi(A) = 0 \wedge \pi_1 = p$ *or*

$S2'$: $\Phi(A) < 0 \wedge \pi_1..r.. \in \sigma(A)$ *or*

$S3'$: $\Phi(A) \leq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge r.. \in \sigma(A)$ *or*

$S4'$: $r..l \in \sigma(A)$.

Up to now we have fixed one point p either on g_1 or on g_2 and have described how one can define a unique valid arc that connects the rightmost point on the other gate reachable from p. In the following we want to discuss, how one can find a valid tight arc $A = (p, \cdot, q)$ such that q is the rightmost point on g_2 reachable from g_1 and p is the rightmost point on g_1 reachable from q.

Definition 8.4 (Rightmost Arc/Leftmost Arc)**.** *Given a feasible constellation* (g_1, g_2, L, R)*. Let* q *be the rightmost point on* g_2 *reachable from* g_1 *and let* p *be the rightmost point on* g_1

that reaches q, then we call the tight arc $A = (p, \cdot, q)$ rightmost arc *(with respect to the direction (g_1, g_2)).*

Analogously we define the leftmost arc *of a feasible constellation.*

Having this definition the following lemma generalizes Lemma 8.4 and Lemma 8.5:

Lemma 8.6 (Variable Starting Point)**.**
Given a feasible constellation $(g_1 = (\pi_1, \omega_1), g_2 = (\pi_2, \omega_2), L, R)$, such that g_2 is reachable from g_1 with respect to the obstacles. Further there is no right obstacle clashing the left boundary arc of g_1 and g_2. Let $A = (p, \cdot, q)$ be a tight arc, then the following statement holds:

A is the rightmost arc if and only if

$R1:\ \Phi(A) = 0 \wedge \pi_1..\pi_2 \in \sigma(A)$ *or*

$R2:\ \Phi(A) < 0 \wedge \pi_1..r..\pi_2 \in \sigma(A)$ *or*

$R3:\ \Phi(A) \leq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge r..\pi_2 \in \sigma(A)$ *or*

$R4:\ \Phi(A) \leq 0 \wedge ((A \parallel g_1 \wedge ..l..r.. \in \sigma(A)) \vee (A \parallel g_2 \wedge l..r.. \in \sigma(A)))$ *or*

$R5:\ r..l..r \in \sigma(A)$.

Proof. "⇒": We assume that $A = (p, \cdot, q)$ is the rightmost arc with respect to (g_1, g_2, L, R). By Definition 8.4 the arc A is a tight arc and q is the rightmost point reachable from p and vise versa. Therefore by Lemma 8.4 and by Lemma 8.5 at least one of the conditions $S1$-$S4$ and at least one of the conditions $S1'$-$S4'$ is true for A. In Table 8.1 we present the different combinations. For the most combinations one can directly see which of the conditions $R1$-$R5$ must follow. For example if $S1$ and $S1'$ are true for A (first group, first line in Table 8.1), then A is a straight line that starts at π_1 and ends at π_2. Consequently A is of type $R1$. Some of the combinations cannot occur at the same time, so for example $S2$ and $S1'$ (first group, second line in Table 8.1). For other combinations the choice of the obstacles (for example the obstacle is a vertex of a gate or not) is crucial. That cases are indicated by a star ($\star$). In the following we only will discuss those cases. All of them have in common, that $S3$ is true for A, that is, A is clockwise oriented, touches g_1 or g_2 and touches a right obstacle r_i, which is not necessarily different from π_2.

Assume that further $S1'$ is true, then A is a straight line starting at π_1. If $r_i = \pi_2$ then A satisfies $R1$. So let $r_i \neq \pi_2$, that is, A does not end at π_2. Without loss of generality we assume that r_i is the last right obstacle touching A. Then a left obstacle must touch A before r_i touches A, because otherwise we can find a valid arc A' going from a point p' on g_1 to a point q' on g_2, such that p' lies to the left of p and q' lies to the right of q and A' intersects A after r_i has touched A (see Figure 8.3a). We only have to set the distance between p' and p and the distance between q and q' appropriate small in order to achieve that A' is valid. On that account A touches first a right obstacle (at π_1), then a left obstacle and finally a right obstacle. Thus, A is of type $R5$.

Now assume that $S2'$ is true, then A is clockwise oriented, starts at π_1 and touches a right obstacle r_j, that must be different from π_2. If $r_i = \pi_2$ then by the previous argumentation, we know that A is of type $R2$. So assume that $r_i \neq \pi_2$, that is, A ends to the left of π_2. We can argue as in the previous case, that is, for the same reasons there must be a left obstacle touching A, before A is touched by the last right obstacle. Again A is of type $R5$.

Finally assume that $S3'$ is true, then (apart from the properties concluded by $S3$) A touches a right obstacle r_j, which is not necessarily different to π_1. If $r_i = \pi_2$, then $R3$ is true. So assume that $r_i \neq \pi_2$ so that A ends to the left of π_2. We can argue in the same way as above, that there must be a left obstacle touching A, before the last right obstacle touches A. If $r_j = \pi_1$, then A touches first a right obstacle, then a left obstacle and finally a right obstacle. Consequently, A is of type $R5$. If $r_j \neq \pi_1$, it can happen that all left obstacles touch A, before any right obstacle has touched A. In this case A satisfies $R4$.

" $\Leftarrow$": Let for the tight arc $A = (p, \cdot, q)$ one of the conditions $R1$-$R5$ be true. We have to show that A is the rightmost arc with respect to (g_1, g_2, L, R), that is, q is the rightmost point reachable from g_1 and p is the rightmost point reachable from q. Obviously, when $R1$ or $R2$ is true for A this property holds.

If $R3$ is true for A, then it also reaches π_2. Therefore q is again the rightmost point reachable from g_1. By $S3'$ of Lemma 8.5 we also can conclude that p is the rightmost point reachable from q.

Now assume that $R4$ is true for A, that is, A is clockwise oriented, touches g_1 or g_2 and first touches a left obstacle and then a right obstacle. Assume by contrary that q is not the rightmost point reachable from g_1, then there must be another valid arc $A' = (p', \cdot, q')$, such that q' lies to the right of q (see Figure 8.3b and 8.3c). In both cases $g_1 \parallel A$ and $g_2 \parallel A$ the arc A' must either start to the left of A or to the right of A and not at p, because due to Lemma 8.4 the point q is the rightmost point reachable from p. If A' starts to the left of A in both cases it also must end to the left of A in order to circumvent the left obstacle as well as the right obstacle. If A' starts to the right of A in the first case ($g_1 \parallel A$), A' intersects the corresponding circle of A twice before it reaches g_2. On that account it must end to the left of A. In the second case ($g_2 \parallel A$) it can only reach g_2 if A' ends to the left of A, because A' must circumvent the same right obstacle as A and A touches g_2. Consequently, q is the rightmost point reachable from g_1. Then we also know from $S3'$ of Lemma 8.5 that p is the rightmost point reachable from q.

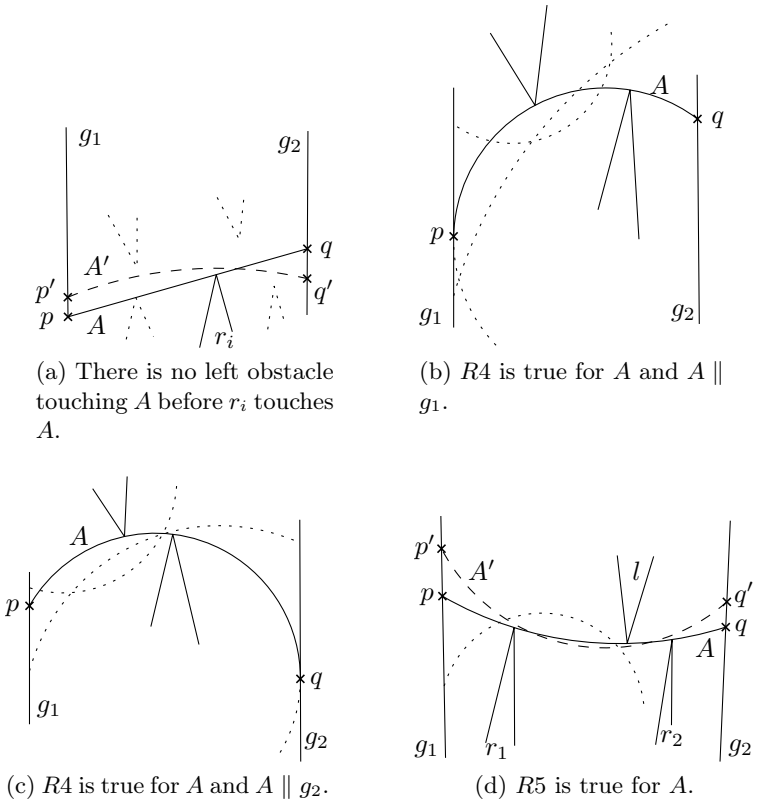
Finally, assume that $R5$ is true for A, that is, A is first touched by a right obstacle r_1, then by a left obstacle l and afterwards again by a right obstacle r_2 (see Figure 8.3d). Assume by contrary that q is not the rightmost point reachable from g_1. Then there is a valid arc $A' = (p', \cdot, q')$, such that q' lies to the right of q. Due to Lemma 8.4 we know that $p' \neq p$.

Going from g_1 to g_2 let A' first lie to the right of A, then it must intersect A before A touches r_1. In order to avoid a clash with l, A' must again intersect A switching from the left hand side to the right hand side of A. But then it cannot circumvent r_2, since it cannot intersect A anymore. On that account A' must first lie on the left hand side of A. In order to avoid a clash with l, A' must intersect A before l touches A. Afterwards it must intersect again A, switching from the right hand side to the left hand side of A, so that it circumvents r_2. But then A' must remain to left of A. Consequently q' cannot lie to the right of q and q therefore must be the rightmost reachable point from g_1. Then we also know from $S4'$ of Lemma 8.5 that p is the rightmost point reachable from q. □

Due to symmetries we can formulate the same lemma for leftmost arcs:

Lemma 8.7 (Variable Starting Point (Symmetric Version))**.** *Given a feasible constellation $(g_1 = (\pi_1, \omega_1), g_2 = (\pi_2, \omega_2), L, R)$, such that g_2 is reachable from g_1 with respect to the obstacles. Further there is no left obstacle clashing the right boundary arc of g_1 and g_2. Let $A = (p, \cdot, q)$ be a tight arc, then the following statement holds:*

A is the leftmost arc if and only if



(a) There is no left obstacle touching A before r_i touches A.

(b) $R4$ is true for A and $A \parallel g_1$.

(c) $R4$ is true for A and $A \parallel g_2$.

(d) $R5$ is true for A.

Figure 8.3: Illustration of the proof for Lemma 8.6.

$L1:\ \Phi(A) = 0 \wedge \omega_1..\omega_2 \in \sigma(A)$ *or*

$L2:\ \Phi(A) > 0 \wedge \omega_1..l..\omega_2 \in \sigma(A)$ *or*

$L3:\ \Phi(A) \geq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge l..\omega_2 \in \sigma(A)$ *or*

$L4:\ \Phi(A) \geq 0 \wedge ((A \parallel g_1 \wedge ..r..l.. \in \sigma(A)) \vee (A \parallel g_2 \wedge r..l.. \in \sigma(A)))$ *or*

$L5:\ l..r..l \in \sigma(A)$.

So far we have introduced lemmas characterizing the right and leftmost arcs in the case that there are no obstacles clashing either the left or right boundary. We still have to discuss the case that there is an obstacle clashing a boundary. Again due to symmetries we only consider the left boundary clashed by a right obstacle.

Lemma 8.8. *Given a feasible constellation $(g_1 = (\pi_1, \omega_1), g_2 = (\pi_2, \omega_2), L, R)$, such that g_2 is reachable from g_1 and there is a right obstacle $r \in R$ clashing the left boundary of g_1 and g_2.*

Then there is no valid arc with respect to (g_1, g_2, L, R).

Proof. We only consider the feasible constellation $(g_1, g_2, \emptyset, r)$ containing the right obstacle r which clashes the left boundary, because if no valid arc exists for $(g_1, g_2, \emptyset, r)$ then no valid arc for (g_1, g_2, L, R) exists. Assume the contrary, that is, there is a valid arc A with respect to $(g_1, g_2, \emptyset, r)$ (see Figure 8.4). Further, let A' denote the left boundary. Since A' connects

	Lemma 8.4		Lemma 8.5	Lemma 8.6
$S1$	$\Phi(A) = 0 \wedge \pi_2 = q$	$S1'$	$\Phi(A) = 0 \wedge \pi_1 = p$	$R1$
		$S2'$	$\Phi(A) < 0 \wedge \pi_1..r.. \in \sigma(A)$	$-$
		$S3'$	$\Phi(A) \leq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge r.. \in \sigma(A)$	$R3$
		$S4'$	$r..l \in \sigma(A)$	$R5$
$S2$	$\Phi(A) < 0 \wedge ..r..\pi_2 \in \sigma(A)$	$S1'$	$\Phi(A) = 0 \wedge \pi_1 = p$	$-$
		$S2'$	$\Phi(A) < 0 \wedge \pi_1..r.. \in \sigma(A)$	$R2$
		$S3'$	$\Phi(A) \leq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge r.. \in \sigma(A)$	$R3$
		$S4'$	$r..l \in \sigma(A)$	$R5$
$S3$	$\Phi(A) \leq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge ..r \in \sigma(A)$	$S1'$	$\Phi(A) = 0 \wedge \pi_1 = p$	$\star$
		$S2'$	$\Phi(A) < 0 \wedge \pi_1..r.. \in \sigma(A)$	$\star$
		$S3'$	$\Phi(A) \leq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge r.. \in \sigma(A)$	$\star$
		$S4'$	$r..l \in \sigma(A)$	$R5$
$S4$	$l..r \in \sigma(A)$	$S1'$	$\Phi(A) = 0 \wedge \pi_1 = p$	$R5$
		$S2'$	$\Phi(A) < 0 \wedge \pi_1..r.. \in \sigma(A)$	$R5$
		$S3'$	$\Phi(A) \leq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge r.. \in \sigma(A)$	$R5$
		$S4'$	$r..l \in \sigma(A)$	$R5$

Table 8.1: Possible combinations of Lemma 8.4 and Lemma 8.5. A star ($\star$) in the last column means that the result is ambiguous (see proof of Lemma 8.6).

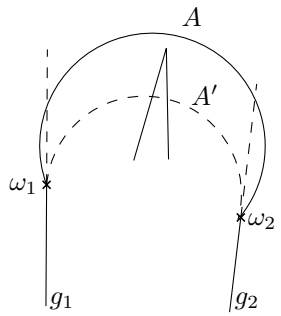


Figure 8.4: Illustration of the proof for Lemma 8.8.

ω_1 with ω_2, we know that A also must connect ω_1 with ω_2. In order to circumvent r, it must completely lie to the left of A', but then it either starts or ends wrongly at g_1 respective g_2. □

It remains to show that we actually can compute right and leftmost arcs. Consider the case $R4$, and assume that we are given an arc A such that $R4$ is true for A. Consequently, there is a left obstacle l touching A then a right obstacle r touching A and A touches g_1 or g_2. In order to be able to compute A we have to assure that there is only a finite set of arcs satisfying the same properties as A with respect to the same obstacle l, r and the gates g_1 and g_2. The next lemma formulates this more precisely:

Lemma 8.9. *Given a feasible constellation $(g_1 = (\pi_1, \omega_1), g_2 = (\pi_2, \omega_2), L, R)$, an arbitrary left obstacle l and two arbitrary right obstacles r and r', then the following statements hold:*

1. *There is at most one valid arc A that satisfies $R1$.*
2. *There is at most one valid arc A that satisfies $R2$ such that it touches r.*
3. *There is at most one valid arc A that satisfies $R3$ such that it touches r.*
4. *There is at most one valid arc A that satisfies $R4$ such that it first touches l and then r.*
5. *There is at most one valid arc A that satisfies $R5$ such that it first touches r then l and finally r'.*

Proof.
There is at most one valid arc A that satisfies $R1$: Since $R1$ means that A is a line segment connecting π_1 with π_2 we can directly conclude the claim.

There is at most one valid arc A that satisfies $R2$ such that it touches r: Assume the contrary, that is, there is another arc A' satisfying $R2$ with respect to r (see Figure 8.5a). Since A and A' already have two points in common, one arc must lie completely to the left of the other. Without loss of generality let A be that arc. We know from both arcs that they are touched by r which is different to π_1 and π_2, but then r must clash A' in order to reach A.

There is at most one valid arc A that satisfies $R3$ such that it touches r: Assume that there is another arc A' satisfying $R3$ with respect to r. Then A' touches either g_1 or g_2 and ends at π_2. We first show that A and A' cannot touch different gates. For that purpose assume without loss of generality that A touches g_1 and A' touches g_2 and both cross the other gate (see Figure 8.5b). Then we can again argue that A' must lie completely to the left of A, so that r cannot touch A' without clashing A. On that account assume that both touch the same gate. Due to symmetries we only consider the case that A and A' touch g_2 (see Figure 8.5c): Since both have the same direction at g_2, the corresponding circles can only have one common point, namely π_2. But then we again derive the case that one arc lies to the left of the other arc.

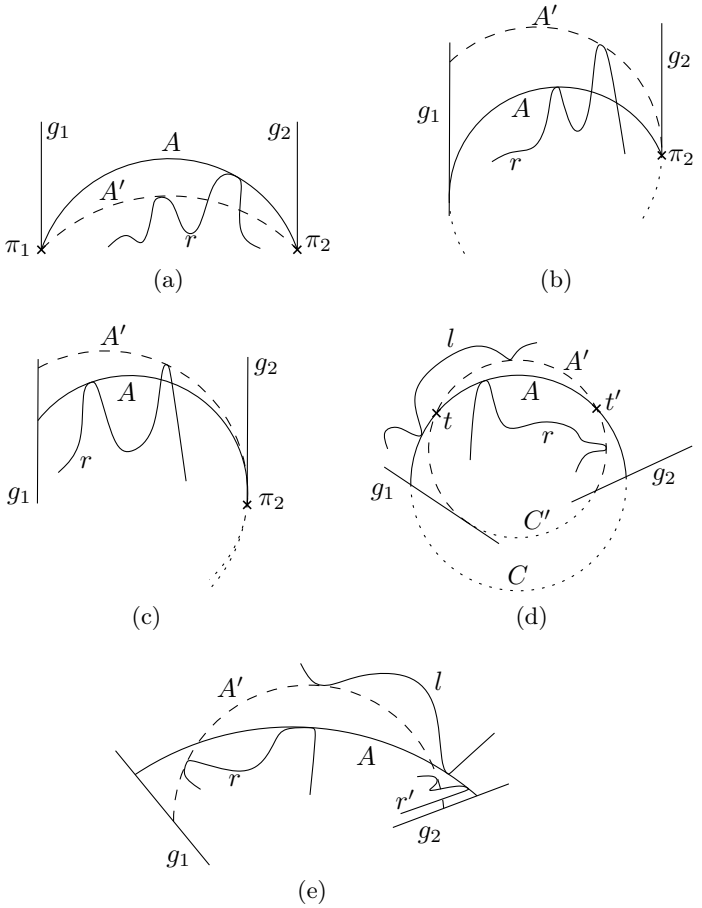


Figure 8.5: Illustration of the proof for Lemma 8.9.

There is at most one valid arc A that satisfies $R4$ such that it touches first l and then r: Again assume that there is another arc A' satisfying $R4$ with respect to l and r (see Figure 8.5d). Note that both arcs are clockwise oriented. Since left and right obstacles are not allowed to have common points (see Definition 7.5), both arcs must intersect twice at points t and t' in order to guarantee that each arc has the possibility to touch first l and then r. Since $A \neq A'$ the arcs must have different radii. Without loss of generality let A be that one with smaller radius. Consider the corresponding circles C and C' of A and A': Then we know that C is contained in C' from t' to t going along C clockwise. In particular that means that both the starting point and the end point of A must be contained in C'. On that account in order that g_1 and g_2 can have common points with A, both gates must cross C'. In particular this means that they also cross A' which is a contradiction to the assumption that at least one of both gates touches A'.

There is at most one valid arc A that satisfies $R5$ such that it touches first r then l and finally r': Let A' be another arc that satisfies $R5$ with respect to r, l and r' (see Figure 8.5e). In order that both arcs have access to the right obstacle they must intersect twice, because otherwise there are too few changes between A and A' such that both cannot touch all obstacles as required. But even if they intersect twice, one of both arcs starts to the

right of the other and consequently ends to the right of the other. Without loss of generality let A be that arc. Since left and right obstacles are not allowed to have common points, the only touching order for A' is that it is first touched by left obstacles, then by right obstacles and finally by left obstacles, which contradicts the assumption. □

Due to that lemma we know that we actually can compute arcs of the different types, assuming that the gates and obstacles are given in such a way that we can compute touching points of arcs with obstacles. In the next section we assemble the results to an algorithm.

8.1.2 Algorithm for Computing Right- and Leftmost Arcs

In this part of the chapter we consider n feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), $\ldots$,(g_{n-1}, g_n, L_n, R_n). Based on Section 8.1.1 we introduce an algorithm solving Problem 8.1, that is, the algorithm computes arcs connecting g_0 with the other gates. For that purpose we extend the definition of Problem 8.1 by introducing left and rightmost arcs. Again due to symmetries we only consider rightmost arcs. The only change we apply to the problem definition is that we do not want to find arbitrary valid arcs but rightmost arcs:

Problem 8.2. *Given n feasible constellations $(g_0, g_1, L_1, R_1), \ldots, (g_{n-1}, g_n, L_n, R_n)$ where $L_i = (l_{i,1}, \ldots, l_{i,k_i})$ and $R_i = (r_{i,1}, \ldots, r_{i,k_i})$ with $k_i \in \mathbb{N}$.*

*Then we want to find a sequence $A_1 = (p_1, \cdot, q_1), \ldots, A_m = (p_m, \cdot, q_m)$ of **rightmost** arcs, such that m is maximized and*

1. *p_i lies on g_0 for all $i \in \{1, \ldots, n\}$ and*
2. *q_i lies on g_i and*
3. *each arc A_i $(1 \leq i \leq m)$ crosses all gates lying between g_0 and g_i and*
4. *for two consecutive gates g_{i-1} and g_i connected by the arc A the sub-arc $A[g_{i-1}, g_i]$ is valid with respect to (g_{i-1}, g_i, L_i, R_i).*

We denote the sequence consisting of the first i sequences R_i by $R_1 \ldots R_i$ and analogously we denote the sequence consisting of the first i sequences L_i by $L_1 \ldots L_i$. Now, we explain a procedure, which we call `connectGatesByRightMostArc`, solving Problem 8.2 (Algorithm 2 gives pseudo-code):

Starting with the pair (g_0, g_1) we compute iteratively for each pair (g_0, g_i) with $1 \leq i \leq n$ a rightmost arc A_i with respect to $(g_0, g_i, L_1 \ldots L_i, R_1 \ldots R_i)$. Obviously, if we cannot find such an arc A_i, that is, g_i is not reachable from g_0, there also cannot be arcs $A_{i+1}, \ldots, A_m$ connecting g_0 with the following gates. We therefore can stop the procedure if we reach a gate g_i that is not reachable from g_0. On that account we also have found a maximum number of gates reachable from g_0.

In order to find a rightmost arc A_i with respect to $(g_0, g_i, L_1 \ldots L_i, R_1 \ldots R_i)$ we first check whether the left boundary of g_0 and g_i is clashed by a right obstacle. Due to Lemma 8.8 there is no valid arc connecting g_0 with g_i. Since we do want that all intermediate gates are passed correctly, we abort at this point the procedure returning $A_1, \ldots, A_{i-1}$.

If there is no right obstacle clashing the left boundary of g_0 and g_i we try to find an arc that satisfies one of the cases stated in Lemma 8.6:

$R1: \ \Phi(A) = 0 \wedge \pi_1..\pi_2 \in \sigma(A)$ or

Algorithm 2: connectGatesByRightMostArc

input : n feasible constellations $(g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), \ldots, (g_{n-1}, g_n, L_n, R_n)$, where $L_i = (l_{i,1}, \ldots, l_{i,k_i})$ and $R_i = (r_{i,1}, \ldots, r_{i,k_i})$ with $k_i \in \mathbb{N}$.

output: A set $\{A_1, \ldots, A_n\}$ of arcs such that for all $i \in \{1, \ldots, n\}$ holds that if g_i is reachable from g_0 then A_i is a rightmost arc with respect to $(g, g_i, L_1 \ldots L_i, R_1 \ldots R_i)$, otherwise $A_i = nil$.

`// In the following` C_i `denotes the corresponding circle of` A_i`.`

$C_0 \leftarrow nil$;
for $i \leftarrow 1$ **to** n **do**
  if *a right obstacle clashes the left boundary of* g_0 *and* g_i **then**
    return $\{A_1, \ldots, A_i, nil, \ldots, nil\}$; `/* set has` n `elements and` $A_i = nil$`. */`
  $A_i \leftarrow$ `computeR1Arc(`g_0, g_i, $(L_1, \ldots, L_i)$, $(R_1, \ldots, R_i)$`)`;
  if $A_i \neq nil$ **then continue**;
  $A_i \leftarrow$ `computeR2Arc(`g_0, g_i, $(L_1, \ldots, L_i)$, $(R_1, \ldots, R_i)$`)`;
  if $A_i \neq nil$ **then continue**;
  $A_i \leftarrow$ `computeR3Arc(`g_0, g_i, $(L_1, \ldots, L_i)$, $(R_1, \ldots, R_i)$`)`;
  if $A_i \neq nil$ **then continue**;
  $A_i \leftarrow$ `computeR4Arc(`g_0, g_i, $(L_1, \ldots, L_i)$, $(R_1, \ldots, R_i)$`)`;
  if $A_i \neq nil$ **then continue**;
  if $i > 1$ *and all obstacles of* L_i *lie to the left of* C_{i-1} *and all obstacles of* R_i *lie to the right of* C_{i-1} **then**
    $A_i \leftarrow$ part of C_{i-1} that goes from g_0 to g_i;
    continue;
  $A_i \leftarrow$ `computeR5Arc(`g_0, g_i, $(L_1, \ldots, L_i)$, $(R_1, \ldots, R_i)$,$\Sigma_{j=1}^{i-1} k_j$`)`;
  if $A_i \neq nil$ **then continue**;
  return $\{A_1, \ldots, A_i, nil, \ldots, nil\}$; `/* set has` n `elements and` $A_i = nil$`. */`
return $\{A_1, \ldots, A_n\}$;

$R2:$ $\Phi(A) < 0 \wedge \pi_1..r..\pi_2 \in \sigma(A)$ or

$R3:$ $\Phi(A) \leq 0 \wedge (A \parallel g_1 \vee A \parallel g_2) \wedge r..\pi_2 \in \sigma(A)$ or

$R4:$ $\Phi(A) \leq 0 \wedge ((A \parallel g_1 \wedge ..l..r.. \in \sigma(A)) \vee (A \parallel g_2 \wedge l..r.. \in \sigma(A)))$ or

$R5:$ $r..l..r \in \sigma(A)$.

We first try to find a valid arc satisfying $R1$. If there is such an arc A_i, we know by Lemma 8.6 that A_i is a rightmost arc and that we can continue with the pair (g_0, g_{i+1}). To that end we introduce the procedure `computeR1Arc` which returns a rightmost arc of type $R1$ if and only if there is such an arc. Otherwise if we can guarantee that there is not such an arc, we consecutively apply the same procedure on $R2$, $R3$ and $R4$. Again we introduce procedures which we use as black-boxes for the moment: `computeR2Arc`, `computeR3Arc` and `computeR4Arc`. If we even cannot find a rightmost arc of type $R4$ we check for $R5$ by first considering the corresponding circle C_{i-1} of the previous arc A_{i-1}. If this arc does not exist

`drawArcR3`($g_1 = (\pi_1, \omega_1)$,r,$g_2 = (\pi_2, \omega_2)$): Returns a clockwise arc A from e_1 through the right obstacle r to e_2 such that A ends at π_2 and A touches e_1 or e_2.
`drawArcR4`($g_1 = (\pi_1, \omega_1)$,l,r,$g_2 = (\pi_2, \omega_2)$): Returns a clockwise arc A that goes from e_1 through l and then through r to e_2 such that A touches e_1 or e_2.
`drawArcR5`($g_1 = (\pi_1, \omega_1)$,r_1,l,r_2,$g_2 = (\pi_2, \omega_2)$): Draws an arc A that goes from r_1, through l to r_2 in that particular order. If this is not possible, it returns *nil*.

Table 8.2: Procedures for drawing arc. The extension of g_1 is denoted by e_1 and the extension of g_2 is denoted by e_2.

because $i = 1$, we continue with the next step. Otherwise we check for all obstacles in L_i and R_i whether they lie on the correct side of A_{i-1}. If the check is positive, we can directly use C_{i-1} to obtain A_i. We just take the part of C_{i-1} that goes from g to g_i. Note that both checks can only be positive if A_{i-1} is of type $R5$. Otherwise we would have found a rightmost arc of type $R1 - R4$ in the previous steps.

If the check is not positive or $i = 1$, we know that at least one obstacle in R_i and L_i must touch A_i (if this arc exists). Since $R5$ is defined as the sequence $r..l..r$, we know that A_i must at least touch a right obstacle in R_i. Providing this information we again call a black-box procedure called `computeR5Arc` which computes a rightmost arc of type $R5$ if it exists.

As already mentioned above, if we cannot even find a valid arc satisfying $R5$ we can stop the whole procedure. In that case we have found a maximum number of rightmost arcs: $A_1, \ldots, A_{i-1}$. In the case that we can find valid arcs for all end gates $g_1, \ldots, g_n$ we obviously also have found a maximum number of rightmost arcs $A_1, \ldots, A_m$ with $m = n$.

Now we explain the single procedures in more detail. For that purpose we first introduce the procedures `clash`, `drawArc` and `check`, which we will need in order to describe the other procedures.

The Procedure `clash`: Given an arc A and an obstacle o, then the procedure checks whether o clashes A as follows: By definition of the problem we only consider obstacles o that belong to a feasible constellation (g_{i-1}, g_i, L_i, R_i) of two consecutive gates g_{i-1} and g_i. We denote the extensions of g_{i-1} and g_i by e_{i-1} and e_i. First we compute the part A_i of A that goes from e_{i-1} to e_i. If this part does not exist, we state that o clashes A. Otherwise we check whether o lies on the correct side of A_i. If o lies on the wrong side of A_i, we again state that o clashes A. In the opposite case we return that o does not clash A.

The Procedure `drawArc`: Table 8.2 lists several variations of this procedure which are used to draw some arc between two gates. Note that all variants of those procedures return an unique arc. The use of this procedure should become more clear in following.

The Procedure `check`: Given a feasible constellation (g_1, g_2, L, R) and an arc A. The procedure checks whether A is valid with respect to (g_1, g_2, L, R) by checking all properties required by Definition 7.7. Obviously the procedure takes $\mathcal{O}(m)$ time, whereat the check whether an obstacles clashes A dominates the time complexity. If A is valid, we say that the result of this procedure is *positive* otherwise *negative*.

The Procedure `computeR1Arc`**:** Given a feasible constellation ($g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2)$, L,R). The procedure returns a rightmost arc of type $R1$ if and only if there is such an arc. For that purpose the procedure draws a line segment A between π_1 and π_2. Afterwards its checks using `clash` whether any obstacle clashes A. In the negative case it returns A, otherwise it returns that there is not such an arc. Obviously the procedure satisfies its postcondition. It takes $\mathcal{O}(m)$ time.

The Procedure `computeR2Arc`**:** Given a feasible constellation ($g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2)$, $L = (l_1, \ldots, l_m)$, $R = (r_1, \ldots, r_m)$). The procedure returns a rightmost arc of type $R2$ if there is such an arc. Based on the arc $A = \overline{\pi_1\pi_2}$ it iteratively considers right obstacles r_i. If r_i clashes with A, we modify A by drawing a new arc $A = (\pi_1, r_i, \pi_2)$. Technically, we have to show that after modifying A, an already considered right obstacle cannot clash A. Since we do the analogous proof for `computeR3Arc`, we resign to do this proof at this place.

After we have considered all right obstacles, we check the resulting arc using `check`. In the negative case we return A, otherwise we state that there is no such arc. Obviously the procedure satisfies its postcondition. It takes $\mathcal{O}(m)$ time.

The Procedure `computeR3Arc`**:** Given a feasible constellation ($g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2)$, $L = (l_1, \ldots, l_m)$, $R = (r_1, \ldots, r_m)$) with extensions e_1 and e_2. We describe a procedure that yields a rightmost arc A of type $R3$ with respect to (g_1, g_2, L, R), if there exists such an arc and otherwise it states that there is not such an arc. Algorithm 3 gives corresponding pseudo-code.

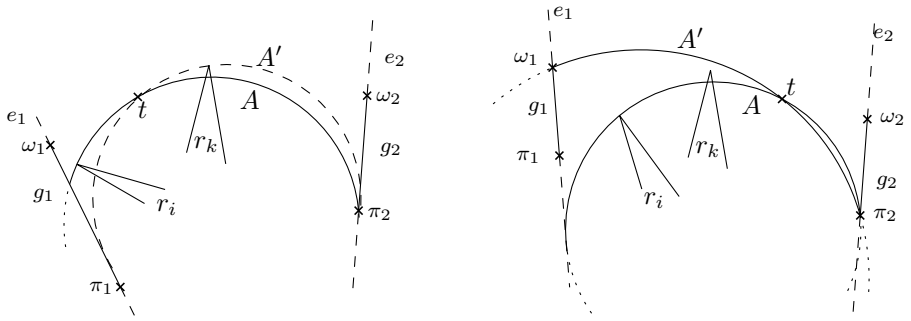
The main idea is that starting only with g_1, g_2 and π_2 we first iteratively add the right obstacles $r_1, \ldots, r_{m-1}$ to that constellation. To that end we start with the arc that corresponds to the line segment $\overline{\pi_1\pi_2}$. Each time when we add a right obstacle r_i, we check whether r_i clashes A. If this is not the case, we consider the next right obstacles. Otherwise we overwrite A by a new clockwise arc that touches r_i and π_2 and that touches e_1 or e_2. As we already have called `computeR1Arc` and `computeR2Arc` we know that there must be at least one right obstacle that clashes the initial arc and from the following lemma we know that for a new arc A the previous right obstacles do not clash with A.

Algorithm 3: computeR3Arc

input : A start gate $g_1 = (\pi_1, \omega_2)$, an end gate $g_2 = (\pi_2, \omega_2)$, left obstacles $L = (l_1, \ldots, l_m)$ and right obstacles $R = (r_1, \ldots, r_m)$.
output: If there is a rightmost arc A with respect to (g_1, g_2, L, R) such that $R3$ is true for A, then A is returned, otherwise *nil* is returned.

```
1
2 A ← π1π2;
3 for i ← 1 to m do
4 |   if clash(A, r_i) then
5 |   |   A ← drawArcR3(g1,ri,g2);/* See Table 8.2                    */
6 if check(g1, g2, A, L, R) then
7 |   return A;
8 return nil;
```

Lemma 8.10. *Given two gates $g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2)$ with extensions e_1, e_2 and right obstacles $R = (r_1, \ldots, r_k)$ such that*



(a) t lies before r_k touches A'. (b) t lies after r_k touches A'.

Figure 8.6: Illustration of the proof for Lemma 8.10.

1. *there is a clockwise arc A that touches π_2 and a right obstacle in $(r_1, \ldots, r_{k-1})$. Further, it touches e_1 or e_2 and clashes with r_k, and*

2. *there is a clockwise arc A' going from e_1 through r_k to π_2 touching e_1 or e_2.*

Then A' does not clash with any obstacle in R.

Proof. Assume the contrary, that is, there is a right obstacle r_i with $1 \leq i \leq k-1$ clashing A' (see Figure 8.6). In order to clash r_i and circumvent r_k the arc A' must intersect A twice: Once at π_2 and once at a point t somewhere between e_1 and e_2. We distinguish two cases where t can lie. But for both cases holds that A and A' cannot arrive at π_2 having the same direction regarding e_2, because otherwise they could not intersect twice.

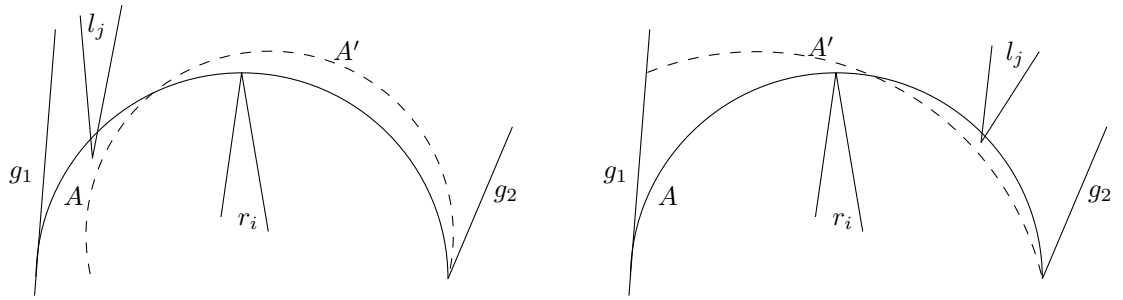
If t lies before r_k touches A' (going from e_1 to e_2, see Figure 8.6a), then A' first lies on the right hand side of A and after t on the left hand side. In order that A' starts from e_1, A must cross e_1. Due to A' ends on the left hand side of A and both are clockwise oriented, A must also cross g_2, because otherwise A would intersect e_2. On that account A can touch neither e_1 nor e_2.

If t lies after r_k touches A' (going from e_1 to e_2, see Figure 8.6b), then A' first lies on the left hand side of A and after t on the right hand side of A. This time A' crosses e_1, because otherwise A could not start from e_1. Since at the end A lies to the right of A', A' must cross e_2, because otherwise A would intersect e_2 (A and A' do not arrive at π_2 having the same direction regarding e_2). □

We continue this procedure until we have considered all right obstacles in R. After traversing all right obstacles A does not clash any right obstacles. This directly follows from the previous lemma.

Note that the resulting A is not necessarily a valid arc. For example it still can clash with left obstacles or reach g_1 or g_2 from the wrong side. We therefore use the function `check` in order to determine whether A is valid. If the check is positive, by Lemma 8.6 we have found a rightmost arc of type $R3$ with respect to (g_1, g_2, L, R). Otherwise we end the procedure stating that there is no rightmost arc of type $R3$ with respect to (g_1, g_2, L, R). From the following lemma we can conclude that this is sound.

Lemma 8.11. *Given a feasible constellation $(g_1{=}(\pi_1, \omega_1), g_2{=}(\pi_2, \omega_2), L = (l_1, \ldots, l_m), R = (r_1, \ldots, r_m))$ with extensions e_1 and e_2, such that there is a clockwise arc A which touches π_2*



(a) t lies before r_k touches A'. (b) t lies after r_k touches A'.

Figure 8.7: Illustration of the proof for Lemma 8.11.

and a right obstacle r_i in R. Further, it touches e_1 or e_2 and does not clash with any right obstacle.

If the result of `check(A, g1, g2, L, R)` *is negative, there is no rightmost arc of type $R3$ with respect to (g_1, g_2, L, R).*

Proof. The function `check` checks for several things, for example whether A reaches g_1 and g_2. But since not reaching g_1 or g_2 is equivalent with clashing obstacles, we only have to consider the case that `check` fails for A because it clashes a left obstacle l_j. Assume that there is a rightmost arc A' of type $R3$. Then A' must circumvent r_i and l_j. There are two cases: Either l_j clashes with A before r_i touches A or vise versa. In both cases there must be an intersection of A and A' between r_i and l_j (see Figure 8.7). Then we can apply the very same arguments as in Lemma 8.10. □

The approach that we have described returns a rightmost arc of type $R3$, if such an arc exists. Otherwise it returns that there is no such arc. Obviously, we consider each right obstacle and each left obstacle at most twice. Thus, the algorithm has a running time of $\mathcal{O}(m)$.

Summarizing this part, we can introduce the following lemma:

Lemma 8.12. *Given two gates $g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2)$, left obstacles $L = (l_1, \ldots, l_m)$ and right obstacles $R = (r_1, \ldots, r_m)$.*

If g_2 is reachable from g_1 by an arc of type $R3$ with respect to (g_1, g_2, L, R), then the procedure `computeR3Arc` *returns a rightmost arc of type $R3$ with respect to (g_1, g_2, L, R), otherwise it returns that this connection is not possible. To that end it uses $\mathcal{O}(m)$ time and $\mathcal{O}(m)$ storage.*

The Procedure `computeR4Arc`**:** Within this procedure (see Algorithm 4) we mainly call the recursive procedure `computeR4ArcRecursive` that returns a clockwise arc A touching a left obstacle l_s and a right obstacle r_t with $1 \leq s \leq t \leq m$ such that

1. A touches e_1 or e_2 and
2. A touches first l_s and then r_t and
3. all clashes of left obstacles l_i with A occur after A touches r_t and
4. all clashes of right obstacles r_i with A occur before A touches l_s.

Later on we explain how `computeR4ArcRecursive` exactly works, but for the beginning assume that A is given. The four properties do not imply that A is a valid arc: It can be that A does not reach g_1 or g_2 or from the wrong side or that it clashes obstacles before l_s or after r_t. In order to check for this, we apply the procedure `check` (Algorithm 4 illustrates `computeR4Arc`). If the result of `check` is positive we know due to Lemma 8.6 that A is the rightmost arc with respect to $(g_0, g_i, L_1 \dots L_i, R_1 \dots R_i)$, otherwise we abort the procedure returning that there is no rightmost arc with respect to $(g_0, g_i, L_1 \dots L_i, R_1 \dots R_i)$. The following lemma shows that this is sound.

Lemma 8.13. *Given a feasible constellation $(g_1{=}(\pi_1, \omega_1), g_2{=}(\pi_2, \omega_2), L = (l_1, \dots, l_m), R = (r_1, \dots, r_m))$ with extensions e_1 and e_2. If there exists an clockwise arc A touching a left obstacle l_s and a right obstacle r_t with $1 \leq s \leq t \leq m$, such that*

1. *A touches e_1 or e_2, and*
2. *A touches first l_s and then r_t, and*
3. *all clashes of left obstacles l_i with A occur after A touches r_t, and*
4. *all clashes of right obstacles r_i with A occur before A touches l_s, and*
5. *there exists a left or right obstacle clashing A,*

then there is no rightmost arc of type $R4$ with respect to (g_1, g_2, L, R).

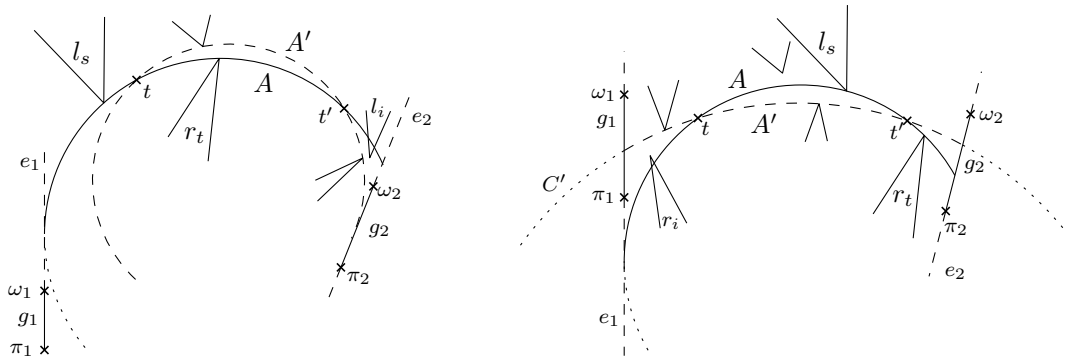
Proof. Assume the contrary, that is, there is a clockwise arc A as described in the lemma and there is a rightmost arc A' of type $R4$ with respect to (g_1, g_2, L, R). Obviously A' must also circumvent the obstacles l_s and r_t.

Assume that A clashes with a left obstacle l_i after it touches r_t (see Figure 8.8a). Since A' is valid, it must circumvent l_i still being clockwise oriented. To that end it must first intersect A between the point where A touches l_s and the point where A touches r_t. We denote this point by t. Then it must intersect A a second time between the point where A touches r_t and the end point of A. We denote that point by t'. Since both arcs are clockwise oriented, we know that the part of A' from g_1 to t must be completely contained in the corresponding circle C of A. The same applies on the part of A' that goes from t' to g_2. Since A only touches e_1 or e_2 and the radius of A' is strict less than the radius of A according to the arguments stated before, A' cannot reach one of both gates.

Assume that A clashes with a right obstacle r_i before it touches l_s (see Figure 8.8b). Again we can argue that there must be two intersection points t and t' between A and A' in order that A' can circumvent first r_i, then l_s and finally r_t. This time the part of A that goes from e_1 to t and the part of A that goes from t' to g_2 must completely lie in the corresponding circle C' of A'. But then e_1 or e_2 can only touch A if g_1 and g_2 cross A', which is a contradiction to the assumption that A' is of type $R4$. □

Since we call `computeR4ArcRecursive` only once and `check` needs $\mathcal{O}(m)$ time assuming that `computeR4ArcRecursive` needs $\mathcal{O}(m)$ time yields a total running time of $\mathcal{O}(m)$.

The Procedure `computeR4ArcRecursive`: We assume that we are given a feasible constellation $(g_1 = (\pi_1, \omega_1), g_2 = (\pi_2, \omega_2), L = (l_1, \dots, l_m), R = (r_1, \dots, r_m))$, whereat we denote the the extensions of g_1 and g_2 by e_1 and e_2. The procedure yields a clockwise arc A touching a left obstacle l_s and a right obstacle r_t with $1 \leq s \leq t \leq m$ such that



(a) A left obstacle l_i clashes A.

(b) A right obstacle r_i clashes A.

Figure 8.8: Illustration of the proof for Lemma 8.13.

1. A touches e_1 or e_2 and
2. A touches first l_s and then r_t and
3. all clashes of left obstacles l_i with A occur after A touches r_t and
4. all clashes of right obstacles r_i with A occur before A touches l_s.

We describe a recursive approach (Algorithm 5 illustrates the procedure): First we draw an arc A that touches l_1 and r_m in that particular order. Further, we require that A touches e_1 or e_2. Since we do not care about the validity of the arc we always can draw such an arc (by using `drawArcR4`). If $m = 1$ we have reached the end of the recursion, that means we return A.

Algorithm 4: computeR4Arc

input : A feasible constellation
$(g_1 = (\pi_1, \omega_1), g_2 = (\pi_2, \omega_2), L = (l_1, \ldots, l_m), R = (r_1, \ldots, r_m))$.

output: If there is a rightmost arc A with respect to (g_1, g_2, L, R) such that $R3$ is true for A, then A is returned, otherwise nil is returned.

```
1 A ← computeR4ArcRecursive(g_0, g_i, L, R);
2 if check(g_1, g_2, A_i, L, R) then return A;
3 return nil;
```

If $m > 1$ it can be that A is clashed by obstacles. In order to resolve such clashes, we go from l_1 to l_m and from r_m to r_1 through the obstacles in an alternating way, that is, when we have considered a left obstacle, the next obstacle is a right one and vise versa. We denote the index of the currently considered left obstacle by i and analogously we denote the index of the currently considered right obstacle by j. We start with $i := 1$ and $j := m$. In the k-th alternating step considering a left or right obstacle we check whether that obstacle clashes with A:

If k is odd we consider a left obstacle. To that end we first refresh the current index by setting $i := i + 1$. Afterwards we check whether A clashes with l_i. If this is not the case we continue with the $(k+1)$-th step considering a right obstacle, otherwise we return `computeR4ArcRecursive`$(g_1, g_2, (l_i, \ldots, l_m), (r_i, \ldots, r_m))$.

Algorithm 5: computeR4ArcRecursive

input : A feasible constellation ($g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2)$, $L = (l_1, \ldots, l_m)$, $R = (r_1, \ldots, r_m)$) with extensions e_1 and e_2.

output: A clockwise arc A touching a left obstacle l_s and a right obstacle r_t, such that i.) A touches e_1 or e_2, ii.) A touches first l_s and then r_t, iii.) all clashes of left obstacles l_i with A occur after A touches r_t, iv.) all clashes of right obstacles r_i with A occur before A touches l_s.

```
1  i ← 1, j ← m, k ← 1;
2  A ← drawArcR4(g1,l1,rm,g2);/* See Table 8.2                    */
3  while i < j do
4      if k mod 2 = 1 then
           // Consider left obstacles.
5          i ← i + 1;
6          if clash(A, li) then
7              return computeR4ArcRecursive(g1,g2,(li,...,lm),(ri,...,rm));
8      else
           // Consider right obstacles.
9          j ← j − 1;
10         if clash(A, rj) then
11             return computeR4ArcRecursive(g1,g2,(l1,...,lj),(r1,...,rj));
12     k ← k + 1;
13 return A;
```

If k is even we consider a right obstacle. We proceed analogously: We first refresh the current index by setting $j := j-1$. Then we check whether A clashes with r_j. If this is not the case, we continue with the $(k+1)$-th step, otherwise we return `computeR4ArcRecursive(`g_1,g_2, $(l_1, \ldots, l_j)$, $(r_1, \ldots, r_j)$`)`.

We stop the alternating procedure if $i = j$ and return A. The correctness of the algorithm follows from the next lemma:

Lemma 8.14. *Given a feasible constellation* (g_1=(π_1, ω_1), g_2=(π_2, ω_2), $L = (l_1, \ldots, l_m)$, $R = (r_1, \ldots, r_m)$) *with extensions* e_1 *and* e_2.

The procedure `computeR4ArcRecursive` *computes a clockwise arc* A *touching a left obstacle* l_s *and a right obstacle* r_t *with* $1 \leq s \leq t \leq m$, *such that*

1. A *touches* e_1 *or* e_2 *and*
2. A *touches first* l_s *and then* r_t *and*
3. *all clashes of left obstacles* l_i *with* A *occur after* A *touches* r_t *and*
4. *all clashes of right obstacles* r_i *with* A *occur before* A *touches* l_s.

Proof. We do an induction over m. Let $m = 1$, that is, $L = (l_1)$ and $R = (r_1)$. By the definition of the algorithm we first draw the arc A, such that A goes from l_1 through r_1 touching g_1. Obviously $i = 1 = j$ and the algorithm returns A. Note that A is not valid, but it satisfies the required properties.

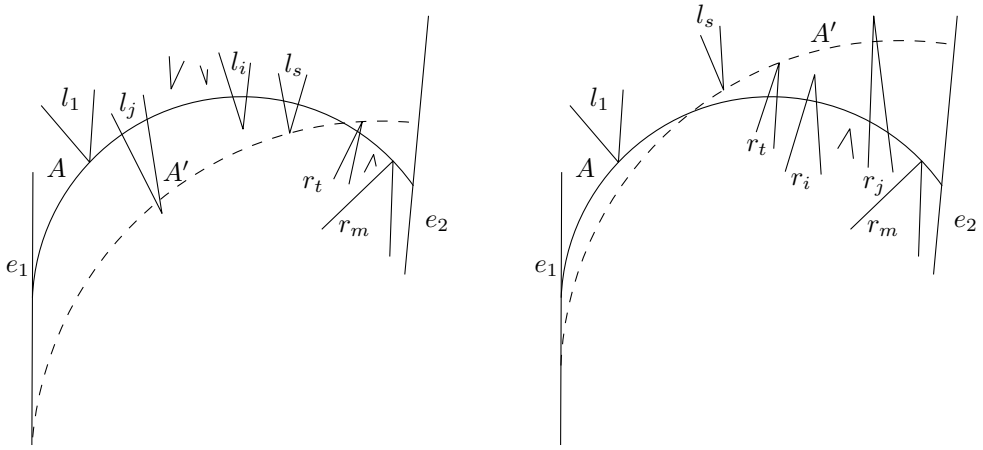


Figure 8.9: Illustration of the proof for Lemma 8.14.

So assume that we have shown the correctness for sequences of obstacles having the size $m-1$ or less. Consider the case that the sequences have size m: $L = (l_1, \ldots, l_m)$ and $R = (r_1, \ldots, r_m)$. Again the method first draws an arc A that touches first l_1 and then r_m. Further, A touches e_1 or e_2. Then the procedure searches for the first obstacle clashing A considering the left and right obstacles in an alternating way. If we cannot find any obstacle clashing A, we return A because the properties are met for A. If there is a clash, we distinguish the cases whether a left or a right obstacle clashing A is found first.

Let $l_i \in L$ be the first obstacle clashing A, then we call `computeR4ArcRecursive` (g_1,g_2, $(l_i, \ldots, l_m)$, $(r_i, \ldots, r_m)$) (see Figure 8.9a) gaining an arc A'. We denote that arc by A'. By induction we know that A' satisfies the required properties with respect to $(l_i, \ldots, l_m)$ and $(r_i, \ldots, r_m)$. In particular A' touches a left obstacle l_s and a right obstacle r_t with $i \leq s \leq t \leq m$. We have to show that $(l_1, \ldots, l_{i-1})$ do not clash A' before A' touches r_t. (Note that $(r_1, \ldots, r_{i-1})$ are allowed to clash A'). Assume that there is a left obstacle l_j with $1 \leq j \leq i-1$ clashing A' before A' touches r_t. First, we show that the clash must occur before l_s touches A'. Assume that the clash occurs after l_s touches A', then by Definition 7.6 there is an arc A'' going from e_1 to l_s then to l_j and finally to e_2. Since $j < s$ this contradicts that (g_1, g_2, L, R) is a feasible constellation. On that account the clash of l_j must occur before l_s touches A'. In order to circumvent l_i the arc A' must lie on the left hand side of A at least until it touches l_s. But then l_j must also clash A in order to clash A', which is a contradiction to $j < i$ and l_i is the first left obstacle clashing A.

Due to symmetries we can analogously argue for the case that $r_j \in R$ is the first right obstacle clashing A (see Figure 8.9b). □

It remains to analyze the running time and the space requirement:

Lemma 8.15. *Given two gates $g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2)$, left obstacles $L = (l_1, \ldots, l_m)$ and right obstacles $R = (r_1, \ldots, r_m)$.*

The function `computeR4ArcRecursive` uses $\mathcal{O}(m)$ time and $\mathcal{O}(m)$ storage.

Proof. In order to analyze the running time of `computeR4ArcRecursive` we estimate the total number $T(m)$ of all loop cycles in `computeR4ArcRecursive` (see Algorithm 4), that is, $T(m)$

also counts the cycles used by the recursive calls. We will show that $T(m)$ has an upper bound of $2m$ by using induction. Let k be a counter of the loop of `computeR4ArcRecursive` that is increased after every cycle by one starting with one. Obviously, calling `computeR4ArcRecursive` recursively has the effect that the loop stops.

If $m = 1$ we can easily show the base case of the induction. Assume therefore that for all sequences of obstacles with less than m obstacles the claim holds. Obviously, if there are no clashes with obstacles, the loop needs m cycles because each time we either increase i or decrease j. So assume that there is an obstacle o that clashes with the given arc A after k cycles.

If k is odd, o is a left obstacle. According to this we call `computeR4ArcRecursive`(g_1, g_2, $(l_i, \ldots, l_m)$, $(r_i, \ldots, r_m)$), where $i = \frac{k-1}{2} + 2$. By induction we know that the running time of that call consumes at most $2 \cdot (m - \frac{k-1}{2} - 2)$ cycles. On that account we know

$$T(m) \leq k + 2 \cdot \left(m - \frac{k-1}{2} - 2 \right) = 2m - 3 \leq 2m$$

If k is even, o is a right obstacle. Then we call `computeR4ArcRecursive`(g_1,g_2, $(l_1, \ldots, l_j)$, $(r_1, \ldots, r_j)$), where $j = m - (\frac{k}{2} - 1)$. By induction we know that the running time of that call consumes at most $2 \cdot (m - \frac{k}{2})$ cycles.

$$T(m) \leq k + 2 \cdot \left(m - \frac{k}{2} \right) = k + 2 \cdot m - k = 2m$$

Since each cycle of the loop takes $\mathcal{O}(1)$ time if we conceptually ignore the recursive calls we obtain a running time of $O(m)$ in total for the loop.

The space requirement follows directly from the fact that we can store both kinds of obstacles in lists.

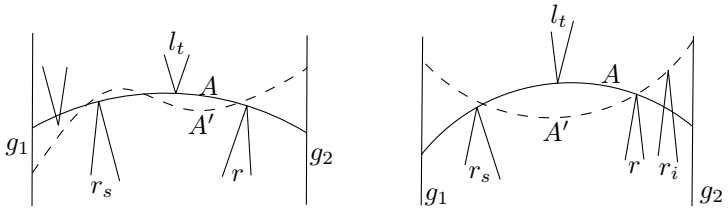
□

The Procedure `computeR5Arc`: We assume that we are given a feasible constellation ($g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2), L = (l_1, \ldots, l_m)$, $R = (r_1, \ldots, r_m)$) and a natural number $k > 1$, such that if there is a rightmost arc A of type $R5$ connecting g_1 with g_2, then there is a right obstacle r_i with $k \leq i \leq m$ touching A. Further, there are no rightmost arcs of type $R1$-$R4$ for that constellation. Note that this also implies that $m > 1$, because otherwise we can find a rightmost arc of type $R1$.

In order to compute a valid arc of type $R5$, we introduce `computeR5ArcRecursive` that computes for each right obstacle r_i with $k \leq i \leq m$ an arc A touching a right obstacle r_s and a left obstacle l_t such that

1. A touches first r_s, then l_t and finally r_i and
2. all clashes of left obstacles l_k with A occur before A touches r_s or after A touches r_i and
3. all clashes of right obstacles r_k with A occur after A touches r_i.

Likewise to $R4$ the three properties do not imply that A is a valid arc. We again use the procedure `check` in order to check for this. In the positive case due to Lemma 8.6 we have found a rightmost arc. Consequently, we can return A. If the check was negative, we consider the next right obstacle (The whole procedure is illustrated in Algorithm 6). The following lemma shows that we do not miss a rightmost arc.



(a) A left obstacle l_i clashes A.

(b) A right obstacle r_i clashes A.

Figure 8.10: Illustration of the proof for Lemma 8.16.

Lemma 8.16. *Given a feasible constellation* $(g_1 = (\pi_1, \omega_1),\ g_2 = (\pi_2, \omega_2), L = (l_1, \ldots, l_m),$ $R = (r_1, \ldots, r_m))$*, with extensions* e_1 *and* e_2*. Further, a right obstacle* r *is given with* $r \neq r_1$*. If there exists a clockwise arc* A *touching a right obstacle* r_s*, a left obstacle* l_t *with* $1 \leq s \leq t \leq m$*, such that*

1. *A touches first r_s, then l_t and finally r and*
2. *all clashes of left obstacles l_k with A occur before A touches r_s or after A touches r and*
3. *all clashes of right obstacles r_k with A occur after A touches r.*
4. *there exists an obstacle clashing A.*

then there is no rightmost arc A' of type $R5$ with respect to (g_1, g_2, L, R) such that A' touches r.

Proof. Assume the contrary, that is, there is a clockwise arc A as described in the lemma and there is a rightmost arc A' of type $R5$ with respect to (g_1, g_2, L, R) such that A' touches r. According to 4. there is a left or right obstacle clashing A.

We first assume that a left obstacle l_i clashes A before r_s touches A or after r touches A (see Figure 8.10a). In both cases A' must circumvent l_i, r_s, l_t and r. But to that end it must be *S-shaped*.

So assume that a right obstacle r_i clashes A (see Figure 8.10b). This must take place after r has touched A. In order to circumvent r_s, l_t, r and r_i the arc A' must first lie to the left of A, then to the right of A and finally to the left of A such that one of both intersection lies between r_s and l_t, and the second one between l_t and r (r touches A'). But then it is not possible to find obstacles touching A' such that $r..l..r \in \sigma(A')$. The arc A prohibits A' from touching obstacles in that particular order. □

If we have considered all right obstacles r_i with $k \leq i \leq m$, but we could not find a rightmost arc of type $R5$ touching one of those r_i we abort the whole procedure stating that there is no rightmost arc of type $R5$ with respect to (g_0, g_i, L, R). This is sound because we assume that if there is a rightmost arc of type $R5$ then it touches a right obstacle r_i with $k \leq i \leq m$.
Since the procedure `computeR4ArcRecursive` is called at most k times assuming that it uses $\mathcal{O}(m)$ time yields a total running time in $\mathcal{O}(k \cdot m)$.

The Procedure `computeR5ArcRecursive`: We assume that we are given a feasible constellation $(g_1 = (\pi_1, \omega_1),\ g_2 = (\pi_2, \omega_2),\ L = (l_1, \ldots, l_m),\ R = (r_1, \ldots, r_m))$ and a right obstacle r,

Algorithm 6: computeR5Arc

input : A feasible constellation ($g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2)$, $L = (l_1, \ldots, l_m)$, $R = (r_1, \ldots, r_m)$) and a number $k \in \mathbb{N}$ such that if there is a rightmost arc of type $R5$ with respect to (g_1, g_2, L, R), then a right obstacle r_i with $k \leq i \leq m$ touches that arc. Further, there are no rightmost arcs of type $R1$-$R4$ with respect to (g_1, g_2, L, R).

output: If there is a rightmost arc A with respect to (g_1, g_2, L, R) such that $R3$ is true for A, then A is returned, otherwise *nil* is returned.

```
1  for i ← k to m do
2      A ← computeR5ArcRecursive(g_0, g_i, L_1 ... L_m, R_1 ... R_m, r_i);
3      if A ≠ nil and check(g_1,, g_2, A, (L_1, ..., L_i), (R_1, ..., R_i)) then
4          return A;
5  return nil;
```

Algorithm 7: computeR5ArcRecursive

input : A feasible constellation ($g_1 = (\pi_1, \omega_1)$, $g_2 = (\pi_2, \omega_2)$, $L = (l_1, \ldots, l_m)$, $R = (r_1, \ldots, r_m)$) and a right obstacle r with $r_1 \neq r$.

output: An arc A touching a right obstacle r_s, a left obstacle l_t with $1 \leq s \leq t \leq m$, such that 1. A touches first r_s, then l_t and finally r, 2. all clashes of left obstacles with A occur before A touches r_s or after A touches r, 3. all clashes of right obstacles with A occur after A touches r..

```
1  i ← 1, j ← m, k ← 1;
2  A ← drawArcR5(g_1, r_1, l_m, r, g_2);/* See Table 8.2                    */
3  if A = nil then
4      if i = j then return nil;
5      return computeR5ArcRecursive(g_1, g_2, (l_1, ..., l_{m-1}), (r_1, ..., r_{m-1}), r)
6  while i < j do
7      if k is odd then
8          i ← i + 1;
9          if clash(A, r_i) then
10             return computeR5ArcRecursive(g_1, g_2, (l_i, ..., l_m), (r_i, ..., r_m), r);
11     else
12         j ← j − 1;
13         if clash(A, l_j) then
14             return computeR5ArcRecursive(g_1, g_2, (l_1, ..., l_j), (r_1, ..., r_j), r);
15 return A;
```

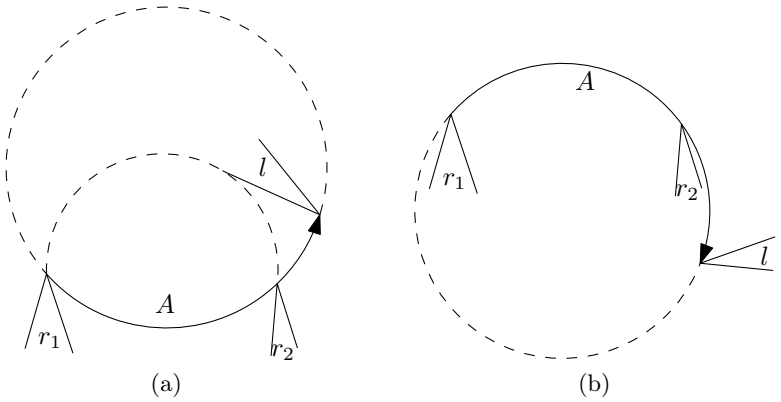


Figure 8.11: Illustration of the procedure `drawArcR5`. Both drawings show the case that it is not possible to draw an arc touching first r_1, then l and finally r_2. A depicts the arc where the orientation of the obstacles is respected, e.g. l lies on the left hand side of A. The dashed bows show arcs which do not respect the orientation of the obstacles.

whereat we denote the extension of g_1 and g_2 by e_1 and e_2 (Algorithm 7 illustrates the procedure). The procedure yields an arc A touching a right obstacle r_s and a left obstacle l_t with $1 \leq s \leq t \leq m$ such that

- A touches first r_s, then l_t and finally r, and
- all clashes of left obstacles l_k with A occur before A touches r_s or after A touches r, and
- all clashes of right obstacles r_k with A occur after A touches r.

The approach we take is very similar to that one computing an arc of type $R4$. We again describe a recursive approach. During the whole procedure we fix r to be an obstacle touching the arc we are looking for. Thus, only r_s and l_t are variable.

We begin with an arc A that touches first r_1 then l_m and finally r (using `drawArcR5`). If we cannot draw such an arc (see Figure 8.11), this happens because we can only find circles touching first r_1 then r and finally l_m (respecting their orientation left and right). In that case we know that l_m cannot be involved in the arc we are looking for. If $m = 1$ we can conclude that there is not a rightmost arc A of type $R5$ touching r. Otherwise we return `computeR5ArcRecursive`$(g_1, g_2, (l_i, \ldots, l_{m-1}), (r_i, \ldots, r_{m-1}), r)$.

If we can draw A, then we iterate through the obstacles in order to resolve clashes. To that end we go from r_1 to r_m and from l_m to l_1 in an alternating way, that is, after we have considered a left obstacle we consider a right obstacle and vise versa. We denote the index of the right obstacle currently considered by i and the index of the left obstacle currently considered by j. We initialize i with 1 and j with m and stop the alternating procedure if $i = j$ returning A. In the k-th step of the algorithm we either consider a right obstacle or a left obstacle:

If k is odd, we consider a right obstacle. For that purpose we first refresh the index i by setting $i := i + 1$. Then we check whether r_i clashes with A. If this is not the case, we continue with the next alternating step $k + 1$. If there is a clash, we call `computeR5ArcRecursive`$(g_1,g_2,(l_i, \ldots, l_m),(r_i, \ldots, r_m), r)$ recursively and return the result.

If k is even, we consider a left obstacle. Accordingly we first refresh the index j by setting $j := j - 1$. Again we check whether l_j clashes with A. In the negative case we continue

with the next alternating step, otherwise we call `computeR5ArcRecursive(`g_1, g_2, $(l_1, \ldots, l_j)$, $(r_1, \ldots, r_j)$, r`)` recursively and return the result.

We first reason about the correctness of the algorithm. To that end we show the following lemma:

Lemma 8.17. *Given a feasible constellation* $(g_1{=}(\pi_1, \omega_1), g_2{=}(\pi_2, \omega_2), L = (l_1, \ldots, l_m), R = (r_1, \ldots, r_m))$ *with extensions* e_1 *and* e_2*. Further, a right obstacle* r *with* $r_1 \neq r$*.*

The call `computeR5ArcRecursive` (g_1,g_2,L,R,r) *yields an arc* A *touching a right obstacle* r_s*, a left obstacle* l_s *and* r *with* $1 \leq s \leq t \leq m$*, such that*

1. A *touches first* r_s*, then* l_t *and finally* r *and*
2. *all clashes of left obstacles* l_k *with* A *occur before* A *touches* r_s *or after* A *touches* r *and*
3. *all clashes of right obstacles* r_k *with* A *occur after* A *touches* r

if such an arc exists, otherwise it states the opposite.

Proof. Analogously to Lemma 8.14 we do an induction over m. Let $m = 1$, that is, $L = (l_1)$ and $R = (r_1)$. By the definition of the algorithm we first draw the arc A, such that A goes from r_1 through l_1 to r. If this is not possible we know that there cannot be an arc we are looking for. Otherwise $i = 1 = j$ and the algorithm returns A. Note that A is not necessarily valid, but it satisfies the required properties.

So assume that for sequences of obstacles with size less than m the claim holds. We now show that it also holds for sequences $L = (l_1, \ldots, l_m)$, $R = (r_1, \ldots, r_m)$ of obstacles with size m. Again we first draw the arc A that goes from r_1 through l_m to r. If we cannot draw A, we return the result of `computeR5ArcRecursive(`g_1,g_2, $(l_i, \ldots, l_{m-1})$, $(r_i, \ldots, r_{m-1})$,r`)`, because we know that l_m is not involved in the arc we are looking for.

If we can draw A, the procedure then searches for the first obstacle clashing A considering the left and right obstacles in an alternating way. If we cannot find such obstacles, we know that A satisfies the requirements. Otherwise we distinguish the cases whether a left or a right obstacle clashing A is found first.

Let $r_i \in R$ be the first obstacle clashing A (see Figure 8.12), then the method returns the arc computed by `computeR5ArcRecursive(`g_1,g_2, $(l_i, \ldots, l_m)$, $(r_i, \ldots, r_m)$,r`)`. We denote that arc by A'. By induction we know that A' satisfies the required properties with respect to $(l_i, \ldots, l_m)$ and $(r_i, \ldots, r_m)$. In particular A' touches a right obstacle r_s and a left obstacle l_t such that $i < s \leq t$. We have to show that $r_1, \ldots, r_{i-1}$ do not clash A'. (Note that $l_1, \ldots, l_{i-1}$ are allowed to clash A'.) Assume that there is an obstacle r_j with $1 \leq j \leq i-1$ clashing A'. We first show that the clash occurs before r_s touches A'. For that purpose assume the contrary, then by Definition 7.6 there is an arc that goes from e_1 to r_s, then to r_j and finally to e_2. Due to $j < s$ this is a contradiction to (g_1, g_2, L, R) is a feasible constellation. Since r_i clashes A but not A', A' must lie to the left of A at least until it touches r_s. On that account r_j must clash with A in order clash with A', but this is a contradiction to $j < i$ and r_i is the first obstacle clashing A.

We can analogously argue for the case that $l_j \in L$ is the first obstacle clashing A. □

Since `computeR4Arc` and `computeR5Arc` do not differ in their structure, we can directly conclude the following lemma about the running time and space requirements of `computeR5Arc`:

Lemma 8.18. *Given two gates* $g_1 = (\pi_1, \omega_1)$*,* $g_2 = (\pi_2, \omega_2)$*, left obstacles* $L = (l_1, \ldots, l_m)$ *and right obstacles* $R = (r_1, \ldots, r_m)$*.*

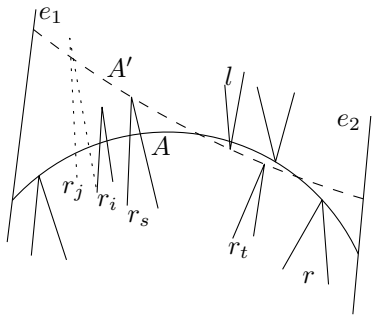


Figure 8.12: Illustration of the proof for Lemma 8.17.

Algorithm 4 uses $\mathcal{O}(m)$ time and $\mathcal{O}(m)$ space.

Proof. See proof of Lemma 8.15. □

Summary
We have shown for `computeR1Arc`, `computeR2Arc`, `computeR3Arc`, `computeR4Arc` and `computeR5Arc` that each of them has a running time in $\mathcal{O}(m)$ when the number of obstacles passed to these methods is denoted by m. Obviously we only consider n pairs of gates: $(g_0, g_1), \ldots, (g_0, g_n)$. On that account we gain a total running time of $\mathcal{O}(n^2 \cdot m)$. Since for two consecutive gates g_{i-1}, g_i there are only a constant number of obstacles we can conclude that $m \in \mathcal{O}(n)$.
Since rightmost arcs and leftmost arcs are symmetric, we can easily modify `connectGatesByRightMostArc` in order to obtain leftmost arcs. We denote the corresponding procedure by `connectGatesByLeftMostArc`. Then we summarize the results in the following theorem:

Theorem 8.1. *We can solve Problem 8.2 and it symmetric variant in $\mathcal{O}(n^2 \cdot m)$ time using $\mathcal{O}(n + m)$ space.*

Further, if between all consecutive pairs of gates only a constant number of obstacles occur then we can solve Problem 8.2 and it symmetric variant in $\mathcal{O}(n^2)$ time using $\mathcal{O}(n)$ space.

Summarizing we have explained how to solve Problem 8.1:

Corollary 8.2. *We can solve Problem 8.1 in $\mathcal{O}(n^2 \cdot m)$ time using $\mathcal{O}(n + m)$ storage.*

Further, if between all consecutive pairs of gates only a constant number of obstacles occur then we can solve Problem 8.2 and it symmetric variant in $\mathcal{O}(n^2)$ time using $\mathcal{O}(n)$ space.

8.2 Computing a Polyarc Through Gates

In this section we introduce some tools for connecting gates with polyarcs of valid arcs using the procedures `connectGatesByRightMostArc` and `connectGatesByLeftMostArc` (see Section 8.1.2) as black-boxes. Further, for the following algorithms we introduce the concept of restricting gates, that is, given a gate g we restrict it to a sub-interval. Conceptually we just move the obstacles of g to the boundaries of the interval we want to restrict to.

The Procedure `restrictGates`: Given a sequence of feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2),..., (g_{n-1}, g_n, L_n, R_n), the first procedure we introduce computes for each gate g_i the points from which g_n can be reached by at least one polyarc of valid arcs. According to Lemma 8.1 the procedure results in $n-1$ closed intervals which later on can be used as restrictions of gates. We go through the gates twice, once from g_0 to g_n and then from g_n to g_0. Traversing the gates firstly, we compute for two consecutive gates g_i and g_{i+1} the reachable points on g_{i+1} regarding g_i and g_{i+1}. If we cannot reach g_{i+1} we also know that g_n is not reachable from g_0. By Lemma 8.1 the reachable points can be described by a closed interval segment I. Before we go over to the next pair g_{i+1} and g_{i+2} we restrict g_{i+1} to I. After reaching g_n we apply the same approach vise versa (Algorithm 8 illustrates the procedure). Obviously, for each restricted gates hold, that from each point, there is a polyarc, such that g_n is reachable. But we also can guarantee that we do not lose points from which g_n is reachable by restricting the gates. Since we only consider each gate twice and we can compute the reachable points between two consecutive gates in $\mathcal{O}(1)$ time, the whole procedure needs $\mathcal{O}(n)$ time.

Algorithm 8: restrictGates

input : Feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2),..., (g_{n-1}, g_n, L_n, R_n)
output: Closed intervals $I_0, \ldots, I_{n-1}$ describing the points on $g_1, \ldots, g_n$ from which g_n is reachable by a polyarc.

```
for i ← 0 to n − 1 do
    A_r ← connectGatesByRightMostArc((g_i, g_{i+1}, L_{i+1}, R_{i+1}));
    A_l ← connectGatesByLeftMostArc((g_i, g_{i+1}, L_{i+1}, R_{i+1}));
    Restrict g_{i+1} to end points of A_r and A_l;
for i ← n down to 1 do
    A_r ← connectGatesByRightMostArc((g_i, g_{i−1}, L_i, R_i));
    A_l ← connectGatesByLeftMostArc((g_i, g_{i−1}, L_i, R_i));
    // Let p_r be the starting point of A_r at g_{i−1} and let p_l be the
       starting point of A_l at g_{i−1}.
    I_i ← [p_r, p_l]
```

Pipes and the Procedure `computePipeSegments`: We assume that we are given a sequence of feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2),..., (g_{n-1}, g_n, L_n, R_n). In order to combine the procedures `connectGatesByRightMostArc` and `connectGatesByLeftMostArc` we introduce pipe segments:

Definition 8.5 (Pipe Segment)**.**
Given the feasible constellations $(g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), \ldots, (g_{n-1}, g_n, L_n, R_n)$. *For two gates* g_i *and* g_j *with* $i < j$ *we call* (I_1, I_2) *a* pipe segment *between* g_i *and* g_j *if* I_1 *describes the reachable interval on* g_i *and* I_2 *describes the reachable interval on* g_j *with respect to* $(g_i,\ g_j,\ L_i \ldots L_j,\ R_i \ldots R_j)$.

We denote the pipe segment between two gates by $P(g_i, g_j)$ *and* I_1 *by* $in(P(g_{i,j}))$ *and* I_2 *by* $out(P(g_i, g_j))$.

Applying Theorem 8.1, for two gates g_i, g_j with $i < j$ we can compute the pipe segments $P(g_i, g_{i+1}), P(g_i, g_{i+2}), \ldots, P(g_i, g_j)$ in $\mathcal{O}((j-i)^2)$ time. We just compute the right and leftmost arc from g_i to g_j in order to obtain I_2 and the right- and leftmost arc from g_j to g_i to

obtain I_1. On that account we also identify a pipe segment $P(g_i, g_j)$ with the corresponding left and rightmost arcs. We denote that procedure by `computePipeSegments`.

In order to connect two gates g_1 and g_n using several pipe segments we introduce the next term:

Definition 8.6 (Valid Chain of Pipe Segments)**.**
Given n feasible constellations $(g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), \ldots, (g_{n-1}, g_n, L_n, R_n)$ and a sequence $p_1, p_2, \ldots p_m$ of pipe segments.

The sequence is called a valid chain *of pipe segments if for all consecutive pipe segments $p_i = P(g_a, g_b)$, $p_{i+1} = P(g_c, g_d)$ it is true that $g_b = g_c$ and $in(p_{i+1}) \subseteq out(p_i)$.*

We also call a valid chain of pipe segments *pipe*.

The Procedure `computePipe`**:** Now we introduce a procedure which computes for a given sequence of feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2),..., (g_{n-1}, g_n, L_n, R_n) a valid chain of pipe segments using a greedy approach. To that end we first restrict the gates $g_0, \ldots, g_n$ using `restrictGates`. If the result is that g_n is not reachable from g_0 we propagate this result to the caller, otherwise we proceed as follows: First we compute all pipe segments starting at g_0 using the procedure `computePipeSegments`. From the resulting pipe segment we take that pipe segment $P(g_0, g_i)$ with greatest index i and call it p_1. If $i = n$ we have reached g_n and we therefore return p_1. Otherwise we restrict g_i to $out(P(g_0, g_i))$ and apply the very same procedure on $(g_i, g_{i+1}, L_{i+1}, R_{i+1}), \ldots, (g_{n-1}, g_n, L_n, R_n)$ in order to obtain a pipe segment p_2. We continue computing pipe segments p_i until we have reached g_n. Due to restricting the gates by `restrictGates` we know that we will reach g_n. Further, by definition of the algorithm we know that $in(p_i) \subseteq out(p_{i+1})$ for two consecutive pipe segments we reach. On that account we obtain a valid chain $p_1, \ldots, p_m$ of pipe segments. (Algorithm 9 illustrates the procedure.) Since for each pipe segment p_i the procedure `computePipeSegments` needs $\mathcal{O}(i^2)$ time and pipe segments do not overlap we can conclude that the procedure has a running time of $\mathcal{O}(n^2)$ using $\mathcal{O}(n)$ space. Then we can use the next procedure in order to translate a valid chain of pipe segments into a polyarc.

The Procedure `computePolyarc`**:** Assume that we are given a sequence of feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), ... , (g_{n-1}, g_n, L_n, R_n) and a corresponding valid chain $p_1, \ldots, p_m$ of pipe segments connecting g_0 with g_n. Using these pipe segments this procedure computes a polyarc $A_1, \ldots, A_m$ of valid arcs connecting g_0 with g_n. For that purpose we start with p_m and choose an arbitrary point in $out(p_m)$ on g_n. From this point we compute a rightmost arc to the start gate of p_m using the procedure `connectGatesByRightMostArc`. Then we apply the same procedure for p_{m-1} and so on until we reach the gate g_0. Since we consider a valid chain of pipe segments we know that we always reach g_0. Again we can argue that for each pipe segment p_i we need $\mathcal{O}((n-i)^2)$ time calling `connectGatesByRightMostArc`. Since pipe segments do not overlap we therefore can conclude a running time in $\mathcal{O}(n^2)$. Further, we know that for each pipe p_i segment we only need $\mathcal{O}(n-i)$ space calling the procedure `connectGatesByRightMostArc`. On that account we obtain space requirement of $\mathcal{O}(n)$.

Up to now, the procedure does not necessarily yield a minimum long chain of arcs. In the next chapter we discuss how such a optimal chain can be obtained.

Algorithm 9: computePipe - Greedy

input : Sequence of feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2),..., (g_{n-1}, g_n, L_n, R_n)

output: Valid chain $p_1, \ldots, p_m$ of pipe segments connecting g_0 with g_n, if this is possible, otherwise nil.

```
1  if restrictGates((g_0, g_1, L_1, R_1),..., (g_{n-1}, g_n, L_n, R_n)) returns 'g_n not reachable'
   then
2      return nil;
3  g ← g_0;
4  i ← 1;
5  while g ≠ g_n do
6      (p'_1 = P(g_0, g_1), ..., p'_m = P(g_0, g_m)) ← computePipeSegments( (g_0, g_1, L_1, R_1),...,
       (g_{n-1}, g_n, L_n, R_n));
7      g ← g_m;
8      p_i ← p'_m;
9      i ← i + 1;
10 return p_1, ..., p_{i-1};
```

9. Advanced Algorithms for the Generalized Corridor

So far we have presented how one can compute polyarcs of valid arcs through given feasible constellations $(g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), \ldots, (g_{n-1}, g_n, L_n, R_n)$, but we have not talked about the optimality of those chains yet. For example `computePipe` does not necessarily return a minimum number of pipe segments applied on given feasible constellations. Consequently, converting the resulting pipe into a polyarc using `computePolyarc` may not yield a minimum number of circular arcs. Further, we only know that the resulting chains are connected, but they are not necessarily smooth. In the following we present algorithms optimizing the length of the resulting chain while also considering smooth solutions.

9.1 Optimizing the Length of a Polyarc Using Predefined Gates

In this section we present how one can solve Problem 7.1, that is, we assume that a sequence $(g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), \ldots, (g_{n-1}, g_n, L_n, R_n)$ of feasible constellations is given and we want to find a minimum long polyarc $(A_1, \ldots, A_m)$ of valid arcs covering the gates. We do not require that the resulting chain is smooth.

Basically, we do an extensive search in oder to find the shortest pipe connecting g_0 with g_n. To that end we introduce a queue Q that contains triples of the form $(g_i, [a, b], p)$, where g_i is a gate, $[a, b] \subseteq [0, 1]$ is an interval on g_i and p is a pipe segment. Later on, p is the segment that is responsible for creating that triple. Further, we denote the currently shortest path by P_{min} and the predecessor of a pipe segment p by $pred(p)$. We initialize Q with $(g_0, [0, 1], \emptyset)$.

As long as Q is not empty we take the next element $(g_i, [a, b], p)$ from Q and try to compute the pipe segments $P(g_i, g_{i+1}), P(g_i, g_{i+2}), \ldots, P(g_i, g_n)$ with respect to the interval $[a, b]$, that is, we restrict g_i to $[a, b]$. We can do this by calling `computePipeSegments`. For each pipe segment $P(g_i, g_j)$ with $j < n$ that we could create, we add $(g_j, out(P(g_i, g_j)), P(g_i, g_j))$ to Q. Further on we set the predecessor $pred(P(g_i, g_j))$ of $P(g_i, g_j)$ to p. If we also could compute $P(g_i, g_n)$, we have reached the last gate: By traversing the predecessors of $P(g_i, g_n)$ we obtain a chain of pipe segments that connect g_0 with g_n. If this chain has fewer segments than P_{min}, we refresh P_{min} using the recently obtained path.

After Q is empty we can apply `computePolyarc` on P_{min} in order to obtain a polyarc from P_{min}. (Algorithm 10 illustrates the procedure.) The following lemma shows the correctness of this procedure as well as its time complexity and space requirement.

Theorem 9.1.
Given n feasible constellations (g_0, g_1, L_1, R_1),(g_1, g_2, L_2, R_2),...,(g_{n-1}, g_n, L_n, R_n). We can obtain a minimum long polyarc of valid arcs covering $g_0, \ldots, g_n$ in $\mathcal{O}(n^4)$ time using $\mathcal{O}(n^3)$ space.

Proof. First we reason about the correctness of the presented approach. We mainly analyze the part in which we create the pipe P_{min}. To that end we show that in each cycle we

Algorithm 10: computePolyarc

input : A sequence $(g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), \ldots, (g_{n-1}, g_n, L_n, R_n)$ of feasible constellations.

output: If g_n is reachable from g_0 then a polyarc of valid arcs and minimum length otherwise nil.

```
1  Q ← a queue initialized with (g_0, [0,1], ∅);
2  P_min ← nil;
3  while Q is not empty do
4      Take next element (g_i, [a,b], p) from Q;
5      Restrict g_i to [a,b];
6      p_1, ..., p_m ← computePipeSegments((g_i, g_{i+1}, L_{i+1}, R_{i+1}), ..., (g_{n-1}, g_n, L_n, R_n));
7      forall p_j ∈ {p_1 ... p_m} do
           // Let g be the end gate of p_i
8          pred(p_j) ← p;
9          if g ≠ g_n then
10             Add (g, out(p_j), p_j) to Q;
11         else
12             Obtain path P from predecessors of p;
13             if P_min = nil or |P| < |P_min| then P_min ← P;
14 if P_min = nil then
15     return P_min;
16 return computePolyarc(P_min,(g_0, g_1, L_1, R_1), ..., (g_{n-1}, g_n, L_n, R_n))
```

only add elements $(g_i, [a, b], p)$ to Q where p is the end segment of a valid chain starting at g_0. We do an induction over the order of the queue. In the first cycle we take from Q the element $(g_0, [0, 1], \emptyset)$. Then we try to compute $P(g_0, g_1), P(g_0, g_2), \ldots, P(g_0, g_n)$ and for each pipe segment $P(g_0, g_i)$ that we could obtain we add $(g_i, out(P(g_0, g_i)), P(g_0, g_i))$ to Q. Obviously, each pipe segment $P(g_0, g_i)$ is a valid chain containing only one element starting at g_0.

So assume that we extract an element $(g_i, [a, b], p)$, then we try to compute $P(g_i, g_{i+1}), P(g_i, g_{i+2}), \ldots, P(g_i, g_n)$ with respect to the interval $[a, b]$. For each pipe segment p' that we could compute we therefore know that $in(p') \subseteq out(p)$. Since p is end point of a valid chain C, the pipe segment p' is end point of the chain $C + p'$.

On that account we only consider valid chains connecting g_0 with g_n when we refresh P_{min}. Since we do an extensive search considering all possibilities we can be sure that we find the shortest valid chain of pipe segments connecting g_0 with g_n.

Now we analyze the running time and space requirement. We first show that the search tree for an extensive search has $\mathcal{O}(n^3)$ nodes:

For gate g_0 we compute n pipe segments $P(g_0, g_1), P(g_0, g_2), \ldots, P(g_0, g_n)$ in the worst case. For the next gate g_1 we compute $n - 1$ pipe segments $P(g_1, g_2), P(g_1, g_3), \ldots, P(g_0, g_n)$ in the worst case. Since for g_2 we created the pipe segments $P(g_0, g_2)$ and $P(g_1, g_2)$ we compute $2 \cdot (n - 2)$ pipe segments. Analogously for g_3 we create $3 \cdot (n - 3)$ pipe segments and so forth. On that account in the worst case we create

$$n + \Sigma_{i=1}^{n} i \cdot (n - i) \leq n + n\Sigma_{i=1}^{n} i = \mathcal{O}(n^3)$$

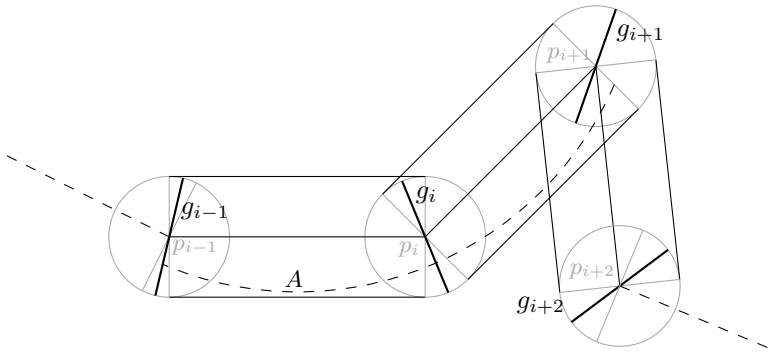


Figure 9.1: Illustration of how to choose gates. Although the arc A is contained in the corridor it does not cross g_{i+1}.

pipe segments. Since for two gates g_i, g_j with $i < j$ we can compute the pipe segments $P(g_i, g_{i+1}), P(g_i, g_{i+2}), \ldots, P(g_i, g_j)$ in $\mathcal{O}((j-i)^2)$ time, we know that we need amortized $\mathcal{O}(n)$ time per pipe segment. Consequently we need $\mathcal{O}(n^4)$ time and $\mathcal{O}(n^3)$ storage. □

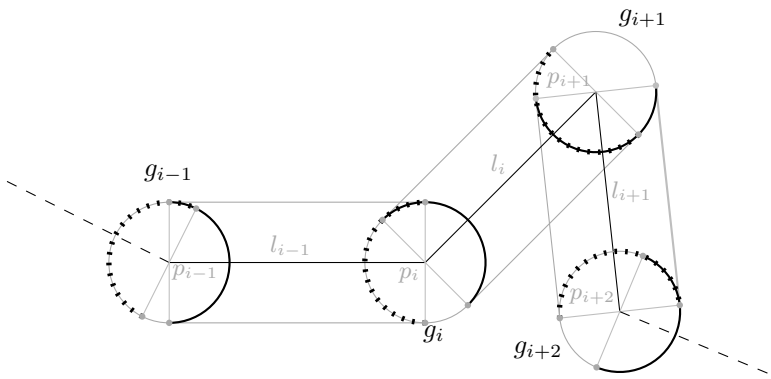
Remark: Since we do not have to store all pipe segments for the whole processing time we also can reduce space requirements, but we let the detailed analysis of this optimization remain open.

For this algorithm we have assumed that fixed gates are given. We have not explained yet how we can gain such gates. Looking back where we came from we have some corridor consisting of corridor segments (see Chapter 5). Each of those segments begins and ends with a circle which we have called connecting circle, because they are the common part of two segments. The idea is to choose for every connecting circle a secant being a gate that must be crossed by the polyarc we are looking for. Since the center of such a connecting circle is also end and start point of two line segments l_1 and l_2 that belong to the polygonal chain we want to cover, the bisectors of the orthogonal lines regarding those line segments lend themselves to be the gates (see g_i in Figure 9.1). Depending on the angle between l_1 and l_2 we also suggest to take the bisector of l_1 and l_2 (see g_{i+1} Figure 9.1). Independently how we choose the gates, we can always find arcs that lie within the corridor without iterating through all gates (see A in Figure 9.1). On that account using gates yields only a approximative solution. Still, the suggested optimization is a generalization of the approach presented in [DRS08]. Summarizing we can state that the presented optimization

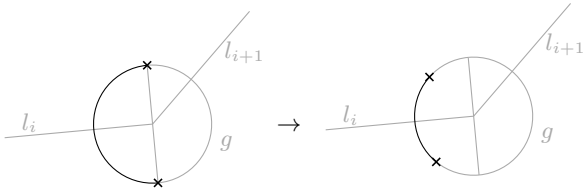
- yields a not necessarily smooth polyarc of optimal length regarding predefined gates, and
- yields an approximation solution if we consider all possibilities how gates can be chosen, whereat we can not estimate the quality factor, and
- need $\mathcal{O}(n^4)$ time and $\mathcal{O}(n^3)$ space.

9.2 Generalization of Gates

In this section we generalize gates in order to compensate the drawbacks of the previous approach. To that end we assume that we are given a polygonal chain $P = (l_1, \ldots, l_n)$ and the



(a) The circles illustrate the single gates, whereat the incoming gate is depicted by a dotted line and the outgoing gate is depicted by a black continuous line.



(b) Illustrates the restriction of a sub-gate.

Figure 9.2: Illustration of generalized gates.

corresponding corridor $\mathcal{C}$ consisting of n corridor segments as defined in Chapter 5. Then we extend the concept of gates. Up to now we only have considered line segments as gates, but we have defined a gate to be a non-self-intersecting curve that satisfies some certain properties. On that account it is very likely that one also can use other geometrical primitives for gates:

We show that circles can also be gates using a little technical trick: The idea is that we soften the restriction that two consecutive arcs we are looking for must end at the very same point on their common gate. From now on we also allow arcs to end at different points on the same gate. On that account we can divide a single gate g into *an incoming gate* $\downarrow g$ and *an outgoing gate* $\uparrow g$ (see Figure 9.2). We represent both sub-gates as sub-segments of g, which are chosen independently from each other. In the case that g is a circle this consequently means that the incoming gate as well as the outgoing gate are sub-arcs of g. As the name already states the incoming gate $\downarrow g$ is responsible for the arcs that arrive at g. Thus, we only consider the back of $\downarrow g$. Analogously, the outgoing gate $\uparrow g$ is responsible for the arcs that leave g and we therefore only consider the front of $\uparrow g$. The gap between an arc A that arrives at g and an arc A' that leaves at g then can bridged by introducing at least one new arc lying in g so that it connects the end point of A with the starting point of A'.

Now, we first show that circular arcs can be used as incoming and outgoing gates and then we explain in more detail how to choose such gates in order to avoid the drawbacks described at the end of the previous section.

Lemma 9.1. *Let $A = (\pi, \cdot, \omega)$ be a circular arc that spans at most the half of its corresponding circle C, then A is a gate.*

Proof. Obviously, the tangential continuations of A do not either intersect each other or A. Then by Lemma 7.1 it follows directly that A is a gate. □

In order to obtain feasible gates for a corridor $\mathcal{C}$ of a polygonal chain P with connecting circles $C_1, \ldots, C_n$ we translate each circle C_i into a gate g. For that we transform the half-circle of C_i which is turned to C_{i-1} into an incoming gate $\downarrow g = (\downarrow \pi, \downarrow \omega)$ such that the outer side of that half-circle is the back of $\downarrow g$ (see Figure 9.2a). Analogously, we translate the half-circle of C_i which is turned to C_{i+1} into an outgoing gate $\uparrow g = (\uparrow \pi, \uparrow \omega)$ such that the inner side of that half-circle is the front of $\uparrow g$. Further, for both sub-gates we introduce obstacles at their endpoints such that the obstacles of $\downarrow g$ belong to the corridor segment (C_{i-1}, C_i) and the obstacles of $\uparrow g$ belong to the corridor segment (C_i, C_{i+1}).

If we have to restrict g to a certain sub-interval, we now do this regarding $\downarrow g$ or $\uparrow g$ (see Figure 9.2b). In both cases we conceptually move the corresponding obstacles of $\downarrow g$ and $\uparrow g$ to the borders of the restricted intervals. Note that we can restrict $\downarrow g$ and $\uparrow g$ independently from each other.

We still have to reason about the validity of arcs. To that end we extend the definition of a valid arc (see Definition 7.7) as follows. We require that a valid arc starting at a gate g_1 and ending at a gate g_2 must particularly start at the outgoing gate $\uparrow g_1$ of g_1 and end at the incoming gate $\uparrow g_2$ of g_2. On the implementation level the drawing of arcs changes a bit. Until now we had to draw arcs touching line segments, from now on we must draw arcs touching line segments and circular arcs.

9.3 Optimizing the Length of a Polyarc Using Generalized Gates

In this part we consider generalized gates as introduced in the previous section and assume that we are given a sequence $(g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), \ldots, (g_{n-1}, g_n, L_n, R_n)$ of feasible constellations. Using `connectGatesByRightMostArc` we greedily connect g_0 with g_n by a sequence $A_1, \ldots, A_m$ of valid arcs, that is, starting at A_1 we choose each circular arc A_i as long as possible. Then we use the destination gate of A_i as the new starting gate of A_{i+1}. On that account the approach works very similar to `computePipe`.

Since arcs do not overlap and for two gates g_i and g_j with $i < j$ holds that from g_j we come at least so far as from g_i, the approach is optimal regarding the length of the sequence. Consequently, there is no other sequence of valid arcs connecting g_0 with g_n having fewer arcs.

However, as already mentioned the so obtained sequence is not a polyarc. Let A_i and A_{i+1} be two consecutive arcs with common gate g, then we can bridge the gap between A_i and A_{i+1} by introducing a line segment l connecting the end point of A_i with the start point of A_{i+1}. Due to the convexity of circles we know that l is contained in g.

Now we reason about the optimality of this approach: Consider a polygonal chain P and its corresponding corridor $\mathcal{C}$ as defined at the beginning of this chapter. Let S be the sequence of feasible constellations we obtain using generalized gates and let $\mathcal{L}$ be the solution which we obtain when we apply the described approach on S. On the other hand let S' be the sequence of feasible constellations using line segments as gates such that the gates are contained in the connecting circles of $\mathcal{C}$ and such that they are optimally chosen. Let $\mathcal{L}'$ be a optimal polyarc regarding S', then $\mathcal{L}$ is 2-approximative with respect to $\mathcal{L}'$.

If we require that the polyarc is smooth we fill the gap between two consecutive arcs A_i and A_{i+1} using at most two arcs which are contained in the corresponding connecting circle. For example we can do this using bi-arcs. For a detailed explanation about bi-arcs see [Bol75].

Since we need $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space for computing all arcs we can summarize the result by the following theorem:

Theorem 9.2.
Given n feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), ..., (g_{n-1}, g_n, L_n, R_n). *We can obtain a polyarc C of valid arcs covering the generalized gates* $g_0, \ldots, g_n$ *in* $\mathcal{O}(n^2)$ *time using* $\mathcal{O}(n)$ *space, such that either*

1. *C is 2-approximative regarding its length when we do not require it to be smooth, or*
2. *C is 3-approximative regarding its length when we do require it to be smooth.*

9.3.1 Covering Circles Using One Line

For the algorithm as presented in Chapter 6 we needed among other things an algorithm for computing a line segment l covering a maximum long prefix of a sequence $C_1, \ldots, C_n$ of circles. We assumed that we use the algorithm as presented in [GHMS91] using $\mathcal{O}(n)$ time. In this section we want to introduce an alternative to that algorithm having the main difference that one can specify more precisely which parts of the circles the line should stab. To that end we again use generalized gates and formalize the problem as follows:

Given a sequence $\mathcal{C} = (C_1, \ldots, C_n)$ of circles, we introduce an algorithm for covering a prefix $C_1, \ldots, C_m$ of $\mathcal{C}$ using one line l such that m is maximized. To that end we consider the polygonal chain $P = (p_1, \ldots, p_n)$ that is induced by the centers of those circles. Based on P we create a corridor using generalized gates as described in Section 9.2. In order to give the user of this procedure the possibility to define the part of a circle where a line can enter, we also expect a corresponding sequence of gates $g_1, \ldots, g_n$ which lie within or on the border of the circles. If they are not specified, we just use the gates as described in Section 9.2.

On that account we obtain (g_1, g_2, L_2, R_2), (g_2, g_3, L_3, R_3), ..., (g_{n-1}, g_n, L_n, R_n) feasible constellations. The idea is that we use the procedures `connectGatesByRightMostArc` and `connectGatesByLeftMostArc` in order to obtain l, but first we show the following lemma:

Lemma 9.2.
Given n feasible constellations (g_0, g_1, L_1, R_1), (g_1, g_2, L_2, R_2), ..., (g_{n-1}, g_n, L_n, R_n), *such that* g_n *is reachable from* g_0.

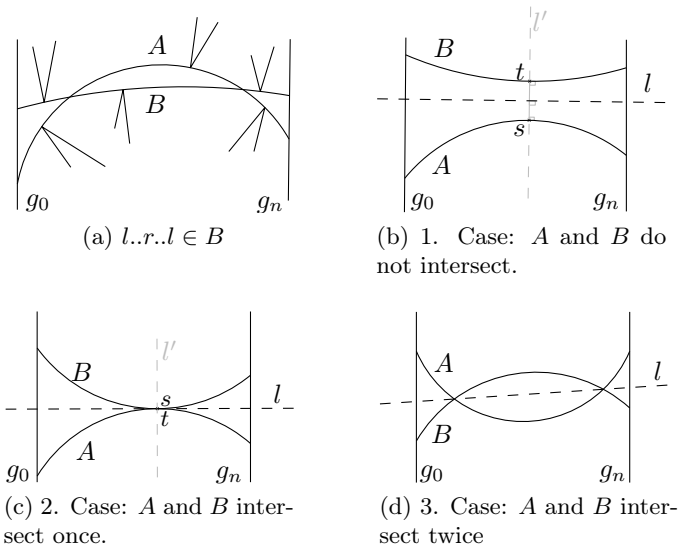
We denote the rightmost arc connecting connecting g_0 *with* g_n *by A and the corresponding leftmost arc by B. Further, we assume that neither A or B is a line.*

Then g_0 *and* g_n *can be connected by a line l with respect to the obstacles if and only if A and B do not have the same orientation.*

Proof. The lemma is well-defined, because if there is a rightmost arc connecting two gates, there is also a leftmost arc. Now we prove both directions:

" $\Rightarrow$ ": We do a proof by contradiction. To that end assume that g_0 and g_n can be connected by a line l with respect to the obstacles, but A and B have the same orientation (see Figure 9.3a). Without loss of generality we assume that both are clockwise oriented. From Lemma 8.7 we then know that $l..r..l \in \sigma(B)$. Since l must circumvent the very same obstacles and B is not a line, l must also be a bow, which is a contradiction to the assumption that l is a line segment. We can argue analogously when both arcs are counterclockwise oriented.

" $\Leftarrow$ ": If A and B do not have the same orientation, we know that A is clockwise oriented and B is counterclockwise oriented. Let l' be the line connecting the centers of A and B. We denote the intersection point of l' and A by s and analogously the intersection point of l' and B by t. We now show how to construct l such that it cannot be clashed by obstacles. To that end we distinguish three cases:



(a) $l..r..l \in B$

(b) 1. Case: A and B do not intersect.

(c) 2. Case: A and B intersect once.

(d) 3. Case: A and B intersect twice

Figure 9.3: Illustration of the proof for Lemma 9.2.

- A and B do not intersect (see Figure 9.3b): We draw the line l through the center of $\overline{st}$ such that l is orthogonal to l'. Since A and B have the same orientation at s respective t as l and l lies between A and B they cannot intersect. According to this l lies to the right of B and to the left of A. Thus, l cannot be clashed by any obstacles.
- A and B intersect once (see Figure 9.3c): We draw the line l through the intersection point of A and B such that l is orthogonal to l'. Since the intersection point of A and B must coincide with s and t, this can be seen a special case of the first case.
- A and B intersect twice (see Figure 9.3d): Let p and p' be the intersection points of both arcs, then we draw l through p and p'. For the whole time going from g_1 to g_n the line l has a sub-arc of A and B to its left and to its right.

Obviously in all three cases l is unique and connects g_0 with g_n.

□

On account of that lemma we introduce the procedure `coverCirclesWithLine`, which computes for a given sequence $\mathcal{C} = (C_1, \ldots, C_n)$ of circles a line l covering a maximum long prefix of $\mathcal{C}$. For that purpose it first calls the procedures `connectGatesByRightMostArc` and `connectGatesByLeftMostArc` on the given feasible constellations obtaining the two sequences

1. $\mathcal{A} = (A_1, \ldots, A_m)$ of rightmost arcs connecting g_1 with its successors and
2. $\mathcal{B} = (B_1, \ldots, B_m)$ of leftmost arcs connecting g_1 with its successors.

In order to do not too much work, we let them run synchronously, that is, first we compute the pairs (A_1, B_1), then (A_2, B_2) and so forth. If we have reached a pair (A_i, B_i) such that both arcs have the same orientation, we abort both procedures. Aborting at this point is correct because we know that a rightmost arc cannot be counterclockwise oriented if the corresponding leftmost arc is clockwise oriented and vise versa. Further, if there is a change in orientation from the $(i-1)$-th pair to the $i-th$ pair, there cannot occur changes anymore.

From Lemma 9.2 we then know that A_{i-1} and B_{i-1} gives us a guarantee that we can connect g_1 with g_i by a line l (Note that due to the choice of indices A_{i-1} and B_{i-1} connect g_1 with g_i). We return the line l as defined in the second part of the proof for Lemma 9.2.

Due to synchronizing `connectGatesByRightMostArc` and `connectGatesByLeftMostArc` we need $\mathcal{O}(m^2)$ time and $\mathcal{O}(n)$ storage which we can summarize in the following theorem:

Theorem 9.3. *Given a sequence $\mathcal{C} = (C_1, \ldots, C_n)$ of circles. Then we can cover a maximum long prefix $C_1, \ldots, C_m$ of $\mathcal{C}$ by a single line l in $\mathcal{O}(m^2)$ time and $\mathcal{O}(n)$ space.*

In particular that means that the algorithm needs $\mathcal{O}(n^2)$ time.

10. Handling Special Cases

So far we have assumed that the connecting circles of the corridor $\mathcal{C}$ of the given polygonal chain P are consecutively disjoint and all of them are of the same radius. In this chapter we also explain how one can apply the approaches discussed so far on corridors for which we allow that the circles overlap. Among other things, there are two typical use cases for which it is necessary to consider overlapping circles:

1. Assuming that the radius of the connecting circles corresponds to the view size of a given map, then a typical use case is that we have zoomed out from the map and see a large part of the map. Then we only want to get some overview of the street modeled by P without following every move of the street. We therefore want to have the possibility to sum up parts of the street.

2. If the radius of the connecting circles does not correspond to the view size, but models the allowed deviation with which we can follow the street represented by P, it still can be reasonable to follow the single moves of P.

For the algorithms based on line stabbing (see Section 6.2) we still can use the old approach, because as presented in [GHMS91] one can still find a line l covering a maximum long prefix of a sequence of arbitrarily placed circles if one only requires that the circles are of the same radius. But in order to find a solution for the first use case, we also suggest another algorithm based on line stabbing. As a second algorithm we present an approach based on gates. Basically, we can summarize both algorithms as follows:

1. **Approach:** We translate P into a polygonal chain P' satisfying Property 6.1 so that we can apply the approach of Section 6.1 on P' in order to obtain a polyarc K covering P. Mainly we work on the intersections of those overlapping circles so that this approach is especially applicable for the first use case. Section 10.1 explains in more detail how one can do this.

2. **Approach:** In Section 10.2 we translate P directly into a polyarc K covering P using gates. Since we do not consider the intersections of the circles as in the first approach this approach is especially applicable for the second use case.

In Section 10.3 we then give a short outlook how to handle circles with different radii.

10.1 Intersection Based Stabbing

In this section we present another algorithm that computes for a given polygonal chain P a simplified polygonal chain P' such that P' satisfies Property 6.1. Then we can use the approach described in Section 6.1 in order to obtain a polyarc.

The idea is to use the intersections of overlapping circles instead of the circles themselves for the line stabbing algorithm as presented in Section 6.2. We compute those intersections such that they are consecutively disjoint. We denote the intersection of a sequence $C_i, C_{i+1}, \ldots, C_j$ by $H_{i,j}$, and describe it by means of circular sub-arcs of those circles, that is, we only consider the rim of the intersection. We call those sub-arcs *edges* of $H_{i,j}$ and the corresponding connection points *vertices* of $H_{i,j}$. Then we can define the problem more formal as:

Problem 10.1. *Given a polygonal chain $P = (p_1, \ldots, p_n)$ with corridor $\mathcal{C}$ and connecting circles $C_1, \ldots, C_n$.*

Then we want to find a minimum/maximum long sequence $\mathcal{H} = (H_{i_1,j_1}, \ldots, H_{i_m,j_m})$ of consecutively disjoint intersections of $C_1, \ldots, C_m$ with $1 = i_1 \leq j_1 < i_2 \leq j_2 < \ldots < i_m \leq j_m$, such that all intersections are not empty.

Both cases searching for a minimum and a maximum long sequence can be reasonable for the first use case as described at beginning of the chapter: If we search for a minimum long sequence we condense P as much as possible, while using a maximum long sequence we condense P only if it is necessary. On that account we obtain in the first case a new polygonal chain that abstracts P more than the one we gain in the second case.

Note that by definition of $H_{i,j}$ we require implicitly that all circles between C_i and C_j are also involved in the intersection. In particular that means the sequence of intersections we are looking for cover the circles with respect to their order. From now on we always consider sequences $\mathcal{H}$ of intersections as described in Problem 10.1 without stating their properties explicitly.

Motivated by the use cases we further assume that the number of overlapping circles is bounded by a number k. More precisely:

Definition 10.1 (Overlapping Bound)**.** *Given a sequence of circles $C_1, \ldots, C_n$. Let H be the intersection of maximum size under all intersections $H_{i,j}$ with $1 \leq i \leq j \leq n$, then the* overlapping bound *of those circles is defined as the size of H.*

Since we assume that P describes a trajectory on a map, it is reasonable to assume that the overlapping bound of P is sub-linear relative to the size of P. For real-world instances it is very likely that this bound is even constant, because it does not seem likely that the overlapping bound of a trajectory should increase with its length: Normally, long trajectories cover a long distance on a map, so that the connecting circles disperse on the map.

In order to compute the intersections as described in Problem 10.1 we first show that it is sufficient to work on the vertices of P and that we do not need to consider the line segments of P. This simplifies the reasoning about those intersections significantly:

Lemma 10.1. *A polygonal chain $P = (p_1, \ldots, p_n)$ can be fully covered by a circle C if and only if C fully covers the convex hull $CH(P)$ of P.*

Proof.
"$\Leftarrow$:" $CH(P)$ is fully contained in C. Assume that there exists a point p on a line segment $l = (p_i, p_{i+1}) \in P$ that is not contained in C. Then at least one of the endpoints of l is not contained within C and consequently not in $CH(P)$. That contradicts that $CH(P)$ is the convex hull of P.
"$\Rightarrow$: P is fully contained in C. That means that in particular the points $p_1 \ldots, p_n$ are contained in C. Consequently, as C is convex all possible connection lines between those points must also be contained in C. In particular the line segments spanning the convex hull $CH(P)$. $\square$

On that account we also can assume that we only have given the circles $C_1, \ldots, C_n$ for solving Problem 10.1. In the next part we first describe how one can compute $H_{i,j}$ for circles $C_i, \ldots, C_j$. Then in the following two parts we explain which of the possible intersections we want to choose. Finally the last part of this section describes how those intersections can be used in order to obtain a simplified polygonal chain P' as described at the beginning of this section.

10.1.1 Computing Intersections of Circles

In [SH76] an divide-and-conquer algorithm is presented that can compute the intersections of the circles $C_1, \ldots, C_n$ in $\mathcal{O}(n \log n)$ time. However, as we are interested in the non-empty intersections $H_{1,1}, \ldots, H_{1,n}$ it is crucial for a good running time to use an iterative approach. The divide-and-conquer approach leads to $\mathcal{O}(n^2 \cdot \log n)$ time. We therefore introduce a simple iterative algorithm with which we can compute those intersections in $\mathcal{O}(n^2)$ time.

More precisely, we describe in this section how one can gain $H_{1,j}$ in such a way that j is maximized while $H_{1,j}$ is still not empty. We apply an iterative approach. For that purpose let H be the current intersection: Starting with $H := C_1$ we iteratively add circles C_i to H: Each time we check whether C_i intersects H which we can obviously do in $O(|H|)$ time, when $|H|$ denotes the size of H. If C_i intersects H we add C_i to H by updating its structure (we can assume that it is modeled as a simple cyclic list of circular arcs). If C_i does not intersect H then we return H. Obviously, H consists of a maximum long prefix of $C_1, \ldots, C_n$ such that it is not empty. The next theorem summarizes the results:

Lemma 10.2. *Given an intersection H of n circles, then we can compute the intersection of H and a circle C in $\mathcal{O}(n)$ time.*

It is very likely that one also can compute the intersection H of i circles with a further circle C faster than in $\mathcal{O}(i)$ time since one can see both H and C as convex polygons based on circular arcs that are to be merged. In [OvL81] an approach is presented how two convex polygons can be intersected efficiently such that the result is again a convex polygon. We let the question remain open whether that approach can be adapted to the problem of computing the intersection of H and C.

10.1.2 A Minimum Long Sequence of Intersections

In this part we explain how one can obtain a minimum long sequence $H_{i_1,j_1}, \ldots, H_{i_m,j_m}$ of consecutively disjoint intersections based on circles $C_1, \ldots, C_n$. To that end we compute the sequence iteratively using a greedy algorithm:

We denote the currently considered intersection by H_i starting with $i = 1$. First we initialize H_i with C_1 in order to start with a non empty intersection. Then we consider the remaining circles $C_2, \ldots, C_n$ successively. Let C_j be the currently considered circle, then we check whether it intersects H_i. If this is the case, we add C_j to H_i, otherwise we create a new intersection H_{i+1} with initial element C_j and increase i by one, so that in the next iteration step we consider the new created intersection. Algorithm 11 shows this approach in more detail. We call this procedure `minIntersectionSequence`, then the following lemma is true:

Lemma 10.3. *Given a sequence $C_1, \ldots, C_n$ of circles with overlapping bound k, then the procedure `minIntersectionSequence` returns a minimum long sequence $H_{i_1,j_1}, \ldots, H_{i_m,j_m}$ of consecutively disjoint intersections with $1 = i_1 \leq j_1 < i_2 \leq j_2 < \ldots < i_m \leq j_m$ such that all intersections are not empty. Further, the procedure needs $\mathcal{O}(n \cdot k)$ time and $\mathcal{O}(n)$ space.*

Algorithm 11: minIntersectionSequence

input : A sequence $C_1, \ldots, C_n$ circles.
output: A minimum long sequence $H_1, \ldots, H_m$ of consecutively disjoint intersections of $C_1, \ldots, C_n$ respecting the order of these circles.

```
1 i ← 1;
2 H_i ← create H_i with first element C_1;
3 for j ← 2 to n do
4     if H_i ∩ C_j = ∅ then
5         H_{i+1} ← create H_{i+1} with first element C_j;
6         i ← i + 1;
7     else
8         H_i ← add C_j to the head of H_i;
9 return H_1, ..., H_i;
```

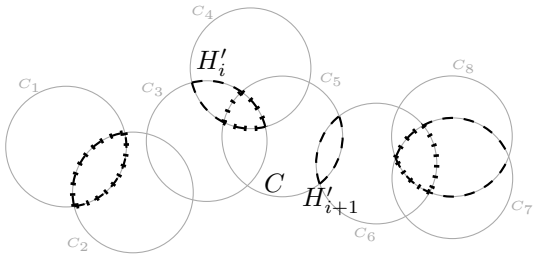
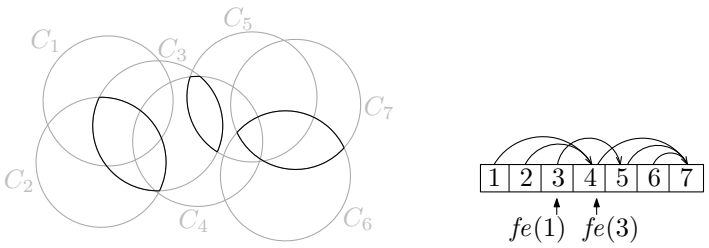


Figure 10.1: Illustration of the proof for Lemma 10.3. The dotted ellipses illustrate the sequence $\mathcal{H}$ and the dashed ellipses illustrate the sequence $\mathcal{H}'$

Proof. It follows directly from the definition of the algorithm that the intersections are consecutively disjoint, because we start to compute the next intersection based on a circle that does not overlap the currently considered intersection. Further, on that account the intersections can not be empty. Due to Lemma 10.2 we need in each iteration step $\mathcal{O}(|H_i|)$ time in order to check whether C_j intersects H_i. Since we know that at most k consecutive circles are involved in the same intersection, we need $\mathcal{O}(k)$ time per iteration step. Thus, we need at most $\mathcal{O}(n \cdot k)$ time in order to compute all intersections. The space requirements follow from the observation that an intersection of i circles can be stored using $\mathcal{O}(i)$ space.

It remains to show that the resulting sequence is minimum regarding its length. To that end assume the contrary, that is, there is a sequence of circles $C_1, \ldots, C_n$ such that the procedure `minIntersectionSequence` yields a sequence $\mathcal{H} = (H_1, \ldots, H_m)$ that has not minimum size. Then there is another sequence $\mathcal{H}' = (H'_1, \ldots, H'_l)$ of consecutively disjoint intersections with less elements. Consequently, there must be an i with $1 \leq i \leq l$ from which on $\mathcal{H}$ and $\mathcal{H}'$ begin to differ. Let H'_i be the first intersection in $\mathcal{H}'$ that is not also contained in $\mathcal{H}$ and let H'_{i+1} be the direct successor of H'_i, then there must be a circle C involved in H'_{i+1} such that C overlaps H'_i and H'_{i+1}, because otherwise H'_i would not be the first intersection that differs (see Figure 10.1). Since H'_i and H'_{i+1} are disjoint but C overlaps both intersections, the intersection $H'_{i+1} - C$ cannot be empty or overlap with H'_i. On that account we can replace H'_i and H'_{i+1} with $H'_i + C$ and $H'_{i+1} - C$ in $\mathcal{H}'$ without gaining an extra intersection. We can apply this approach until $\mathcal{H} = \mathcal{H}'$ which is a contradiction to the assumption that $\mathcal{H}$ and $\mathcal{H}'$



(a) Intersection of circles.

(b) The corresponding relation *lrc*.

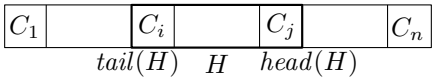
Figure 10.2: Illustration for the last reachable circle.

are of different size. □

10.1.3 A Maximum Long Sequence of Intersections:

In this section we discuss the counterpart of the previous section, that is, given a sequence $C_1, \ldots, C_n$ we want to find a *maximum* long sequence $H_{i_1,j_1}, \ldots, H_{i_m,j_m}$ of consecutively disjoint intersections with $1 = i_1 \leq j_1 < i_2 \leq j_2 < \ldots < i_m \leq j_m$ such that all intersections are not empty. We again apply a kind of greedy algorithm, but this time we change from one intersection to the next one as early as possible.

For the rest of this section we assume that we store the circles $C_1, \ldots, C_n$ in one array in that particular order. Further, we store for an intersection $H_{i,j}$ only the two indices i and j. Hence, the index i is called the *tail* of $H_{i,j}$ which we denote by $tail(H_{i,j})$. Analogously we call j the *head* of $H_{i,j}$ which we denote by $head(H_{i,j})$:



For the sake of readability we directly identify the circles with their indices, that is, for example we write $1, \ldots, n$ instead of $C_1, \ldots, C_n$ and say 'the circles $1, \ldots, n$'.

Let $i, \ldots, j$ be the longest sequence of circles starting at a circle i whose intersection is not empty, then we denote the circle j by $lrc(i)$ (**l**ast **r**eachable **c**ircle of i). Due to Lemma 10.2 and the overlapping bound k we can compute for each circle i the index $lrc(i)$ in $\mathcal{O}(k^2)$ time: We start with an intersection H that only contains the circle i, then we add iteratively the circles $i+1, i+2, \ldots$ to H until it is empty. The last circle j with $H \cap C_j \neq \emptyset$ yields the index $lrc(i)$.

Since we have n circles we can pre-compute *lrc* for all circles in $\mathcal{O}(n \cdot k^2)$ time. Further, we let every circle i directly point to its index $lrc(i)$ so that the request $lrc(i)$ only needs $\mathcal{O}(1)$ time. Figure 10.2 illustrates the definition of *lrc*. Obviously, for a circle j in a given sequence $i, \ldots, j, \ldots, lrc(i)$ it is true that $lrc(j) \geq lrc(i)$.

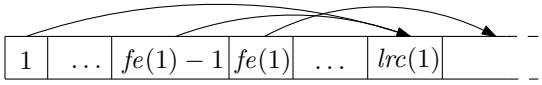
We call the first circle j in $i, \ldots, lrc(i)$ with $lrc(j) > lrc(i)$ the ***f**irst **e**scapee* of i. If such a circle does not exist for i, we call $lrc(i)+1$ the first escapee of i. On that account if $lrc(i) = n$, it also can be that the first escapee is not a circle but only an number. We define that more formal as:

$$fe(i) = \begin{cases} lrc(i)+1 & \text{for all circles } j \text{ within } i, \ldots, lrc(i) \text{ it is true that } lrc(j) = lrc(i) \\ j & j \text{ is the first circle in } i, \ldots, lrc(i) \text{ with } lrc(j) > lrc(i) \end{cases}$$

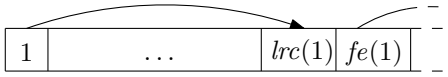
Obviously we can pre-compute fe for every circle i in $\mathcal{O}(k)$ time if we assume that we already have computed lrc for i.

The main idea for computing a maximum long sequence is that beginning with H_1 we choose each intersection as small as possible in order to cover the whole sequence with as many intersections as possible. For the intersection H_1 we can distinguish the following two cases:

1. Case: $fe(1) \leq lrc(1)$



2. Case: $fe(1) > lrc(1)$

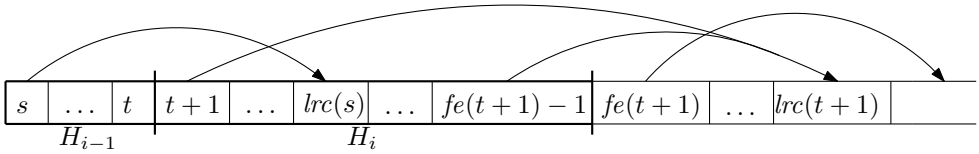


In both cases H_1 must cover at least the circles $1, \ldots, fe(1) - 1$ in order to avoid that the next intersection H_2 overlaps H_1: If we chose H_1 such that it ends before $fe(1) - 1$ at a circle i, H_2 would have to start with circle $i + 1$ for that we know that $i + 1 \leq fe(1) - 1$. Thus, H_2 can only cover circles up to $lrc(1)$, which yields that H_1 and H_2 are not disjoint.

On that account we set $H_1 = (1, \ldots, fe(1) - 1)$. In the case that $fe(1) - 1 = n$ we further know that we have summed up the sequence $C_1, \ldots, C_n$ of circles using a maximum long sequence of consecutively disjoint intersection.

After discussing the base case we explain how to specify the i-th intersection H_i assuming that there is an intersection $H_{i-1} = (s, \ldots, t)$. Obviously, H_i starts with $t+1$ as its first circle. We distinguish the cases $lrc(s) < fe(t + 1) - 1$ and $lrc(s) \geq fe(t + 1) - 1$:

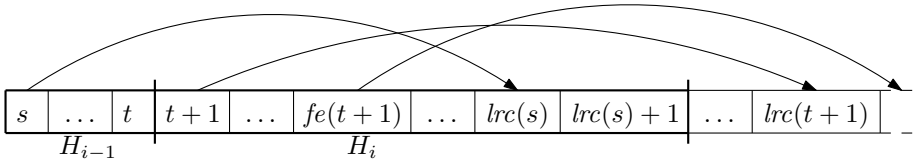
1. Case: $lrc(s) < fe(t + 1) - 1$



We can argue analogously as in the base case that H_i must cover at least the circles $t + 1, \ldots, fe(t + 1) - 1$ in order to avoid that the next intersection H_{i+1} overlaps H_i: If H_{i+1} ended at a circle u with $t + 1 < u < fe(t + 1) - 1$, the next intersection H_{i+1} would have to start at a circle $\leq fe(t + 1) - 1$. Due to the definition of fe, then H_{i+1} must end somewhere between $u + 1, \ldots, lrc(t + 1)$. Thus, H_i and H_{i+1} would intersect.

On that account we set in that case $H_i = (t + 1, fe(t + 1) - 1)$. From the reasoning above it directly follows that H_i is of minimum size with respect to its successors. Since $lrc(s) < fe(t + 1) - 1$ the intersection H_i and H_{i-1} must be disjoint.

2. Case: $lrc(s) \geq fe(t + 1) - 1$



This time H_i must cover at least the circles $t+1, \ldots, lrc(s)+1$ in order to be disjoint with H_{i-1}. The argumentation for that can be done analogously to the previous cases. We therefore set H_i to be the sequence $(t + 1, \ldots, lrc(s) + 1)$. It directly follows that H_i is minimum size with respect to H_{i-1} and that H_i does not intersect H_{i-1}.

In both cases we do not need to consider further intersections H_{i+1}, H_{i+2}, ... if the head

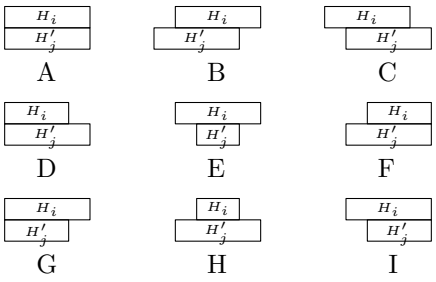


Figure 10.3: Possible types of overlapping intersections.

of H_i is the circle C_n. Finally we can summarize the cases as follows:

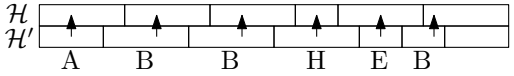
$H_1 := (1, \ldots, fe(1) - 1)$

and for all i with $i > 1$ and $head(H_{i-1}) < n$:

$$H_i := \begin{cases} head(H_{i-1}) + 1, \ldots, fe(tail(H_i)) - 1) & lrc(tail(H_{i-1})) < fe(head(H_{i-1}) + 1) - 1 \\ head(H_{i-1}) + 1, \ldots, lrc(tail(H_{i-1})) + 1) & lrc(tail(H_{i-1})) \geq fe(head(H_{i-1}) + 1) - 1 \end{cases}$$

We denote the procedure that computes these intersections by `maxIntersectionSequence`. That those intersections are consecutively disjoint and cover the circles $C_1, \ldots, C_n$ directly follows from the description above. It remains to show that the resulting sequence $\mathcal{H} = (H_1, \ldots, H_m)$ is maximized. To that end we compare it to another arbitrary sequence $\mathcal{H}' = (H'_1, \ldots, H'_l)$ that covers $C_1, \ldots, C_n$ being consecutively disjoint.

We say that an intersection H of circles *overlaps* an intersection H' of circles, if there is at least one circle in H that is also contained in H'. Figure 10.3 shows all possible overlapping types for two intersections H_i of $\mathcal{H}$ and H'_j of $\mathcal{H}'$. We call those compositions *bricks*. We can decompose $\mathcal{H}$ and $\mathcal{H}'$ into bricks, which not necessarily work out even:

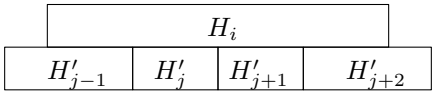


Hence, we assume that the upper layer depicts $\mathcal{H}$ and the lower layer depicts $\mathcal{H}'$. Later on we use bricks in order to show that $\mathcal{H}$ is maximized, but first the next lemma and its corollary state some general properties of disjoint intersections:

Lemma 10.4. *Given a sequence of circles $C_1, \ldots, C_n$ and the sequence $\mathcal{H} = (H_1, \ldots, H_m)$ that is the result of `maxIntersectionSequence` applied on the given circles. Let $\mathcal{H}' = (H'_1, \ldots, H'_l)$ be another arbitrary sequence of consecutively disjoint intersections covering $C_1, \ldots, C_n$, then the following statement is true:*

For each intersection H_i in $\mathcal{H}$ it is true that it can overlap at most three intersections of $\mathcal{H}'$.

Proof. Assume the contrary. Hence, there is an intersection H_i which overlaps more than three intersections of $\mathcal{H}'$. We always can choose four intersections H'_{j-1},H_j, H'_{j+1}, H'_{j+2} of those overlapped intersections such that they occur in a row. Since all four of them have at least one circle in common with H_i, the two intersections H'_j, H'_{j+1} contain only intersections that are also contained by H_i:



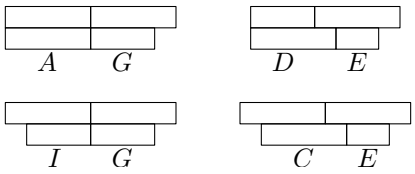


Figure 10.4: Not possible combinations of bricks.

But then H_i testifies to the fact that H'_j and H'_{j+1} cannot be disjoint. □

Corollary 10.1. *If H_i overlaps two or three intersections, then only one of those intersections can be completely overlapped by H_i. In the case that H_i overlaps three intersections, the intersection in the middle is completely overlapped.*

Since the corollary is obvious, we will not state it explicitly each time when we make use of it. We now show that some bricks require other certain bricks as predecessors:

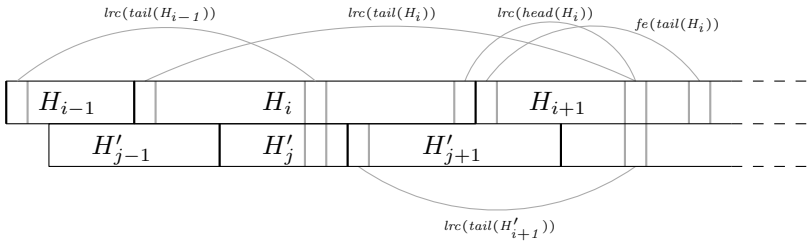
Lemma 10.5. *Given a sequence of circles $C_1, \ldots, C_n$ and the sequence $\mathcal{H} = (H_1, \ldots, H_m)$ that is the result of* `maxIntersectionSequence` *applied on the given circles. Let $\mathcal{H}' = (H'_1, \ldots, H'_l)$ be an arbitrary sequence of consecutively disjoint intersections covering $C_1, \ldots, C_n$, then the following statement is true:*

For two consecutive bricks (H_{i-1}, H'_{j-1}) and (H_i, H'_j) it cannot be that

$$head(H'_j) < head(H_i) \text{ and } tail(H_i) \leq tail(H'_j) \text{ and } tail(H_{i-1}) \leq tail(H'_{j-1})$$

Figure 10.4 illustrates typical cases that are not possible due to that lemma. We show this lemma using a proof by contradiction. To that end remember that for all intersections H the head and tail of H always satisfy $tail(H) \leq head(H)$.

Proof. Assume the contrary. Hence, there are two sequences $\mathcal{H}$ and $\mathcal{H}'$ as described in the lemma, but we can find intersections H_{i-1}, H_i, H'_{j-1} and H'_j such that $head(H'_j) < head(H'_i)$, $tail(H_i) \leq tail(H'_j)$ and $tail(H_{i-1}) \leq tail(H'_{j-1})$. We illustrate the setting as follows:



In order to satisfy the assumption H_i must be of size of at least two while the other intersections must have a size of at least one. Further, it must be true that

$$lrc(tail(H_{i-1})) < head(H'_j), \tag{10.1}$$

because otherwise the intersection $tail(H_{i-1}) \cap \ldots \cap lrc(tail(H_{i-1}))$ would be a witness that the intersections H'_{j-1} and H'_j are not disjoint (The figure above illustrates the extreme case that $lrc(tail(H_{i-1})) = head(H'_j) - 1$). Thus, and because by assumption $head(H'_j) < head(H_i)$

it is true that

$$head(H_i) - head(H'_j) \geq 1 \overset{(10.1)}{\leftrightarrow} \tag{10.2}$$
$$head(H_i) - lrc(tail(H_{i-1})) \geq 2 \leftrightarrow \tag{10.3}$$
$$head(H_i) \geq lrc(tail(H_{i-1})) + 2 \leftrightarrow \tag{10.4}$$
$$head(H_i) > lrc(tail(H_{i-1})) + 1 \tag{10.5}$$

Since H_i has a predecessor the head of H_i is defined distinguishing two cases:

$$lrc(tail(H_{i-1})) < fe(tail(H_i))-1 \Rightarrow head(H_i) = fe(tail(H_i))-1 \tag{10.6}$$
$$lrc(tail(H_{i-1})) \geq fe(tail(H_i))-1 \Rightarrow head(H_i) = lrc(tail(H_{i-1}))+1 \tag{10.7}$$

Due to Inequation (10.5) and the right part of Implication (10.7) we know that $lrc(tail(H_{i-1})) < fe(tail(H_i)) - 1$ must be true and consequently by Implication (10.6) that

$$head(H_i) = fe(tail(H_i)) - 1 \tag{10.8}$$

As by assumption $head(H'_j) < head(H_i)$ there must be an intersection H'_{j+1} with $tail(H'_{j+1}) \leq head(H_i)$. Consequently, $lrc(tail(H'_{j+1})) \leq lrc(head(H_i))$. Then from Equation (10.8) it follows that

$$head(H'_{j+1}) \leq lrc(tail(H'_{j+1})) \leq lrc(head(H_i)) \overset{(10.8)}{=} lrc(fe(tail(H_i)) - 1) \overset{\text{Def. } fe}{=} lrc(tail(H_i)) \tag{10.9}$$

Further, as by assumption $tail(H'_j) \geq tail(H_i)$ the next inequation follows:

$$tail(H_i) \leq tail(H'_j) \leq head(H'_j) < tail(H'_{j+1}) \leq head(H'_{j+1}) \leq lrc(tail(H_i)) \tag{10.10}$$

Consequently, the intersection $tail(H_i) \cap \ldots \cap lrc(tail(H_i))$ proves that the two intersections H'_j and H'_{j+1} are not disjoint, which is a contradiction to the definition of $\mathcal{H}'$. □

From that lemma we can directly conclude the next one, that states more concretely which combinations of bricks are not possible.

Lemma 10.6. *The following sentences are true:*

1. *The predecessor of a brick of type G can only be of type F.*
2. *The predecessor of a brick of type E can only be of type H.*

Proof. We only show the first sentence, because the other can be proven analogously. If a brick b of type G has a predecessor b', then b' must be of type A, F or I due to the shapes of b and b' (see Figure 10.3). But from Lemma 10.5 it directly follows that b' cannot be of type A or I (see Figure 10.4). □

We now show that every brick of type G must have a predecessor:

Lemma 10.7. *Given a sequence of circles $C_1, \ldots, C_n$ and the sequence $\mathcal{H} = (H_1, \ldots, H_m)$ that is the result of* `maxIntersectionSequence` *applied on the given circles. Let $\mathcal{H}' = (H'_1, \ldots, H'_l)$ be an arbitrary sequence of consecutively disjoint intersections covering $C_1, \ldots, C_n$, then the following statement is true:*

For all bricks (H_i, H'_j) of type G it is true that there are previous intersections H_{i-1} and H'_{j-1}.

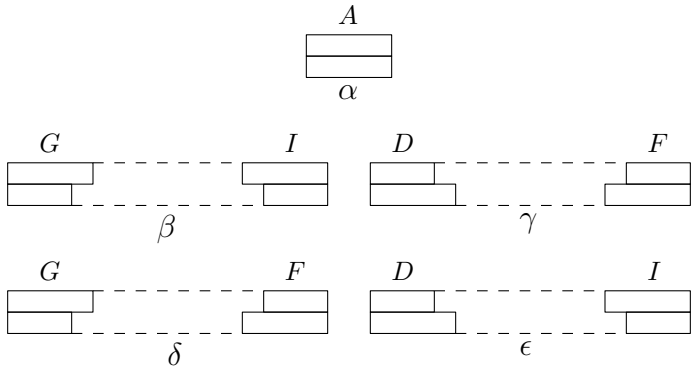
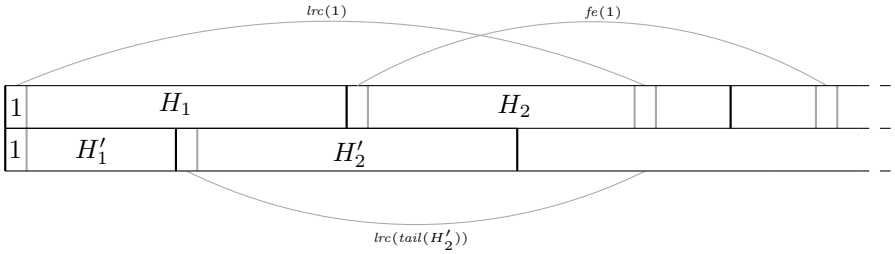


Figure 10.5: Possible types of blocks.

Proof. Assume the contrary. Thus, there are no previous intersections, which means that $i = 1$ and $j = 1$:



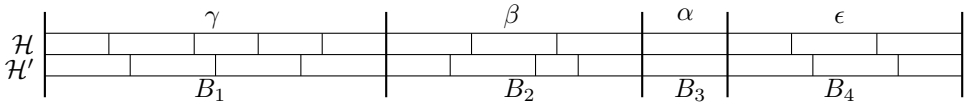
From the definition of H_1 we know that $head(H_1) = fe(1) - 1$. Since $head(H'_1) < head(H_1)$ it must be then true that there is an intersection H'_2 with $tail(H'_2) \leq fe(1) - 1$. Due to the definition of fe it is true that $lrc(tail(H'_2)) \leq lrc(1)$. But then it also must be true that

$$1 \leq tail(H'_1) \leq head(H'_1) < tail(H'_2) \leq head(H'_2) \leq lrc(1)$$

Consequently, the intersections H'_1 and H'_2 must overlap, a contradiction to the definition of $\mathcal{H}'$. □

Now we assemble bricks to *blocks* whose beginning is induced by bricks of type A, D and G and whose end is induced by bricks of type A, F and I. Figure 10.5 shows all possible types of blocks. We always choose blocks as small as possible, that is, bricks of type A, D, F, G or I are only allowed to form the bounds of a block but may not occur within a block. In that sense those blocks are atomic.

Obviously, we always can decompose $\mathcal{H}$ and $\mathcal{H}'$ into a unique sequence $B_1, \ldots, B_s$ of blocks:



The next lemma summarizes some statements about blocks that helps us to prove that $\mathcal{H}$ is maximum long.

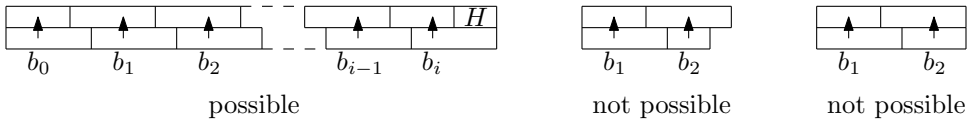
Lemma 10.8. *Given a sequence of circles $C_1, \ldots, C_n$ and the sequence $\mathcal{H} = (H_1, \ldots, H_m)$ that is the result of* `maxIntersectionSequence` *applied on the given circles. Let $\mathcal{H}' = (H'_1, \ldots, H'_l)$ be an arbitrary sequence of consecutively disjoint intersections covering $C_1, \ldots, C_n$ and let $B_1, \ldots, B_s$ the sequence of blocks induced by $\mathcal{H}$ and $\mathcal{H}'$ then the following statements are true:*

1. *The block B_1 cannot be of type β or δ.*
2. *Blocks of type γ always contain one intersection of $\mathcal{H}$ more than of $\mathcal{H}'$.*
3. *Blocks of type β can contain at most one intersection of $\mathcal{H}'$ more than of $\mathcal{H}$.*
4. *The only type that can contain more intersections of $\mathcal{H}'$ than of $\mathcal{H}$ is type β.*
5. *There are at least as many blocks of type γ as of type β.*

Proof. We prove the statements in the given order:

The block B_1 cannot be of type β or δ. Follows directly from Lemma 10.7.

Blocks of type γ always contain one intersection of $\mathcal{H}$ more than of $\mathcal{H}'$. Let B be an arbitrary block of type γ, then we decompose B into bricks $b_0, \ldots, b_i$ starting from its beginning:



By definition of type γ the first brick b_0 must be of type D. According to the shape of that brick, the next brick b_1 must be of type C, E or I. But due to Lemma 10.6 type E can only occur if its predecessor is of type H. So b_1 cannot be of type E. It also cannot be of type I because then the brick would form the end of the block, which is a contradiction to the definition of type γ, namely B must end with a brick of type F. Consequently, b_1 must be of type C. We can analogously argue for the remaining bricks $b_2, \ldots, b_i$ that they must be of type C. However, as B ends with a brick of type F, there must be an intersection H of $\mathcal{H}$ at the end of B that does not belong to any brick $b_0, \ldots, b_i$. Thus, the block B contains one intersection of $\mathcal{H}$ more than of $\mathcal{H}'$.

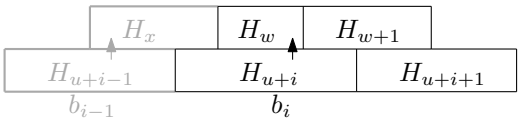
Blocks of type β can contain at most one intersection of $\mathcal{H}'$ more than of $\mathcal{H}$. Let B be an arbitrary block of type β, then we again decompose B into bricks $b_0, \ldots, b_i$ starting from its beginning:

Obviously, at least b_0 and b_1 must exist. As B starts with a brick of type G, due to the shape of G the remaining bricks can only be of type C, B, E, H or I.

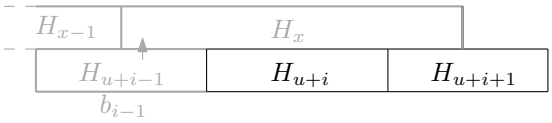
In order to prove the claim, we give an algorithm that describes the decomposition of B into bricks in more detail: To that end let $H_s, \ldots, H_t$ be the intersections in $\mathcal{H}$ that are contained in B in increasing order. Analogously, let $H'_u, \ldots, H'_v$ be the intersections in $\mathcal{H}'$ that are contained in B in increasing order. We iteratively go through the intersections $H'_u, \ldots, H'_v$, defining for each intersection H'_{u+i} with $0 \leq i \leq v - u$ a brick b_i. Each time after we have defined a brick b_i we conceptually delete it from B, so that the intersection involved in b_i are not available for the next bricks.

We define the first brick as $b_0 := (H_s, H'_u)$ and delete it from B. Assume that we have defined the first bricks $b_0, \ldots, b_{i-1}$ for the intersections $H'_u, \ldots, H'_{u+i-1}$ (and have delete them from B) and that there is a further intersection H'_{u+i}. Let H_w be the first intersection that

overlaps H'_{u+i} such that it has not been deleted yet. If this intersection does not exist, we stop the whole procedure, otherwise we set $b_i := (H_w, H'_{u+i})$:



H_w exists.

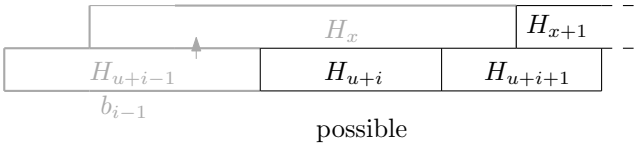


H_w does not exist.

In particular if we cannot find H'_w, the intersection H_{u+i} remains undeleted. Consequently, we obtain a maximum long prefix of B that can be decomposed into bricks $b_0, \ldots, b_{i-1}$. We show that we always reach H'_v decomposing B, that is, in particular we do not necessarily delete H'_v but all previous intersections $H'_u, \ldots, H'_{v-1}$ have been deleted, which means that there is at most one intersection of $\mathcal{H}'$ more contained in B than of $\mathcal{H}$.

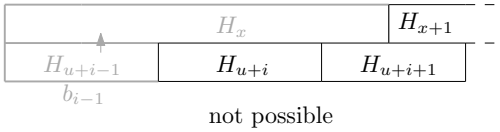
Assume the contrary, that is, we stop in the i-th step considering the intersection H'_{u+i} with $u + i < v$. Let the previous brick be defined as $b_{i-1} = (H_x, H'_{u+i-1})$. The only reason for stopping is that the intersection H_x completely overlaps H'_{u+i} (right figure). Thus, H_x and H'_{u+i} must form a brick of type E: It cannot be of type C because then we have the case as depicted in the left figure, that is, there is an intersection H_w that has not been deleted yet and overlaps H'_{u+i}. It also cannot be of type I because then H'_{u+i} would be the last intersection of B, which means that $u + i = v$.

Further, the brick $b_{i-1} = (H_x, H'_{u+i-1})$ must be of type B:

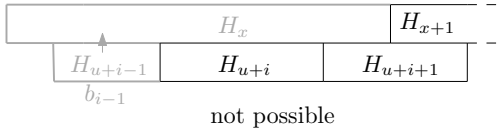


possible

It cannot be of type G or E because then H_x would be a witness that H'_{u+i-1} and H'_{u+i} are not disjoint:

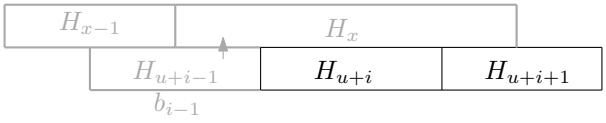


not possible



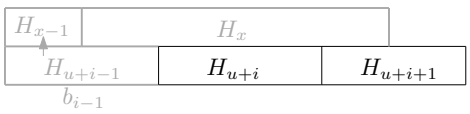
not possible

Moreover, the intersections H_{x-1} and H'_{u+i-1} form a brick of type C:

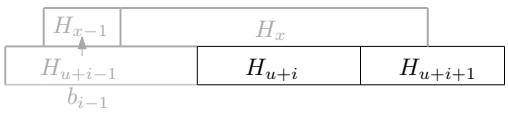


possible

It cannot be of type D or H because then b_{i-1} would be defined as (H_{x-1}, H'_{u+i-1}):



not possible



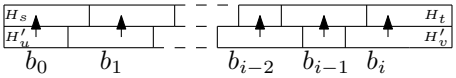
not possible

Obviously, the brick (H_{x-1}, H'_{u+i-1}) is the predecessor of the brick (H_x, H'_{u+i}). But as (H_x, H'_{u+i}) is of type E as argued above and due to Lemma 10.6, it is a contradiction that (H_{x-1}, H'_{u+i-1}) is of type C.

On that account only the last intersection H'_v can remain, without that it is involved in one of the bricks $b_1, \dots, b_{i-1}$. Thus, B has at most one intersection of $\mathcal{H}'$ more than of $\mathcal{H}$.

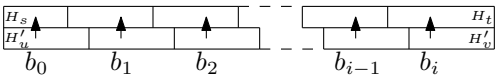
The only type that can contain more intersections of $\mathcal{H}'$ than of $\mathcal{H}$ is type β. Obviously, blocks of type α contain exactly two intersections, one of each sequence $\mathcal{H}$ and $\mathcal{H}'$. Due to the second sentence blocks of type γ contain more intersections of $\mathcal{H}$ than of $\mathcal{H}'$.

For blocks of type δ we can analogously argue as in the previous sentence: We can decompose those blocks into bricks $b_0, \dots, b_i$ such that we always reach the intersection H'_v:



Since by definition of type δ the intersections H'_v and H_t form a brick of type F, we can always guarantee that H_v does not remain, but that we can use H_t for forming a the brick b_i.

For blocks of type ϵ we again decompose those blocks into bricks $b_0, \dots, b_i$. Analogously argued as in the second sentence, the successors $b_1, \dots, b_{i-1}$ of b_0 can only be of type C:

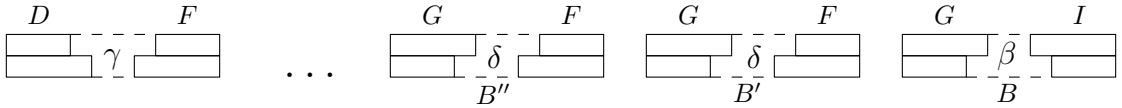


Due to the definition of type ϵ the last two intersections H_t and H'_v of B must form a brick of type I. Since all previous intersections $H'_{u+1}, \dots, H'_{v-1}$ form with some intersection of $\mathcal{H}$ a brick of type C and H'_u and H_s form a brick of type D, the last intersection H_t is not involved in those bricks. On that account H'_v and H_t form the brick $b_i = (H_t, H'_v)$ of type I.

Since there are no other types of blocks, the claim follows. We now show that we can find for each block of β a unique block of type γ, because then we know that $\mathcal{H}$ contains at least as many intersections as $\mathcal{H}'$.

There are at least as many blocks of type γ as of type β. Let B be a block of type β, then from Sentence 1 it directly follows that there is a previous block B'. As B begins with a brick of type G the block B' can only end with a brick of type F (Lemma 10.6). Thus, B' can only be of type γ or δ:

If B' is of type γ we have found a corresponding block for B. So let B' be of type δ. Again, due to Sentence 1 there must be a previous block B'' of B'. Since B' also begins with a brick of type G, the last brick of B'' can only be of type F (Lemma 10.6). Consequently, B'' can only be of type γ or δ as B'. On that account we can go backwards from B only traversing blocks of type δ until we reach a block that is of type γ:



□

Corollary 10.2. *The sequence $\mathcal{H}$ contains at least as many intersections as the sequence $\mathcal{H}'$.*

We summarize the results we have made so far in the following theorem, which directly follows from the previous lemmas and considerations.

Theorem 10.1. *Given a sequence of circles $C_1, \dots, C_n$ with overlapping bound k, then the sequence $\mathcal{H} = (H_1, \dots, H_m)$ covers $C_1, \dots, C_n$ being consecutively disjoint and maximum long, if we define the intersections as follows:*

$H_1 := (1, \ldots, fe(1) - 1)$

for all i *with* $i > 1$ *and* $head(H_{i-1}) < n$*:*

$$H_i := \begin{cases} (head(H_{i-1})) + 1, \ldots, fe(tail(H_i)) - 1) & , lrc(tail(H_{i-1})) < fe(tail(H_i)) - 1 \\ (head(H_{i-1})) + 1, \ldots, lrc(tail(H_{i-1})) + 1) & , lrc(tail(H_{i-1})) \geq fe(tail(H_i)) - 1 \end{cases}$$

Further, we can compute $\mathcal{H}$ *in* $\mathcal{O}(n \cdot k^2)$ *time and* $\mathcal{O}(n)$ *space.*

10.1.4 Using the Intersections

In this section we describe how one can apply the results of the previous sections on a given polygonal chain P in order to obtain a circular arc covering the corridor $\mathcal{C}$ of P. We distinguish two different variants where the second one can be seen as an enhancements of the first one.

1. Variant: Let $C_1, \ldots, C_n$ be the connecting circles of the given corridor $\mathcal{C}$, then we compute based on those circles the sequence $H_1, \ldots, H_m$ of consecutively disjoint intersections using the procedure `minIntersectionSequence` or `maxIntersectionSequence`. To that end we choose for each intersection H_i with $1 \leq i \leq m$ an arbitrary point p'_i in H_i (e.g. the centroid of the convex polygon spanned by the vertices of H_i) and set $P' = (p'_1, \ldots, p'_m)$. Due to the following lemma P' covers all circles $C_1, \ldots, C_n$ with respect to their order:

Lemma 10.9. *Given a polygonal chain* $P = (p_1, \ldots, p_n)$ *with corridor and corresponding connecting circles* $C_1, \ldots, C_n$*, such that the intersection* H *of those circles is not empty. Let* r *be the radius of those circles.*

Then every circle C *with radius* r *and center* c *fully covers* P *if and only if* $c \in H$*.*

Proof. From Lemma 10.1 we know that we only need to consider the vertices of P.
"$\Leftarrow$:" Assume C fully covers P, but c is not located in H. Then there is a vertex p of P with $\|c - p\|_2 > r$, that is, p cannot be covered by C.
"$\Rightarrow$:" c is located in all circles C of r in radius and center p for all $p \in P$. Consequently C covers the centers of these circles. □

Further, each line segment of P' covers at least two disjoint circles, so that P' satisfies Property 6.1. Thus, we can apply the approach of Section 6.1 on P'.

Theorem 10.2. *Given a polygonal chain* P *with not necessarily disjoint connecting circles* $C_1, \ldots, C_n$ *with overlapping bound* k *then we can find in* $\mathcal{O}(k \cdot n)$ *(*$\mathcal{O}(k^2 \cdot n)$*) time using the procedure* `minIntersectionSequence` *(*`maxIntersectionSequence`*) a polyarc* K*.*

Proof. It remains to argue for the running time, which directly follows from the fact that we can compute $\mathcal{H}$ in $\mathcal{O}(n \cdot k)$ or $\mathcal{O}(n \cdot k^2)$ time using the procedure `minIntersectionSequence` or `maxIntersectionSequence` (Lemma 10.3, Theorem 10.1). For the approach of Section 6.1 we only need $\mathcal{O}(n)$ time (Theorem 6.1). □

2. Variant:
We again compute the sequence $\mathcal{H} = (H_1, \ldots, H_m)$ of consecutively disjoint intersections on $C_1, \ldots, C_n$ using `minIntersectionSequence` or `maxIntersectionSequence`: But this time we compute P' such that it has minimum size and minimum number of inflection points regarding all polygonal chains covering $\mathcal{H}$. For that purpose we can use the approach of Section 6.2 only changing the procedure `coverCirclesWithLine`.

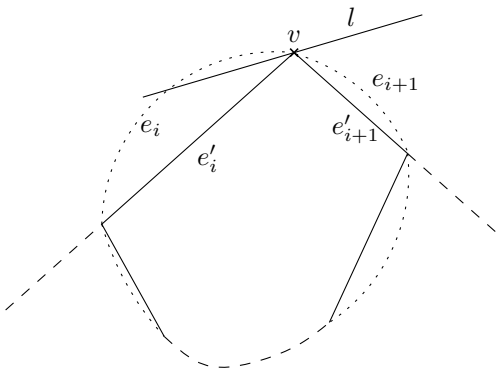


Figure 10.6: Illustration of the proof for Lemma 10.10.

As already mentioned in Section 4.2 the procedure `coverCirclesWithLine` can be easily adapted to more complex objects than circles. In our case those objects are the intersections $\mathcal{H}$. Mainly, one has to explain how inner and outer tangents can be computed. The idea is that we first consider the tangents of the convex polygons that are induced by the vertices of the intersections and then we use those tangents in order to obtain the tangents of the considered intersections:

Let H and H' be two arbitrary intersections of circles such that they have size n_1 and n_2, respectively. We denote the convex polygons induced by the vertices of H and H' by Q and Q', respectively. According to [OvL81, Pre79, KS95] one can compute outer (inner) tangents between Q and Q' in $\mathcal{O}(\log(n_1 + n_2))$ ($\mathcal{O}(\log(n_1) \cdot \log(n_2))$) time. The following lemma shows that we can use those tangents in order to gain corresponding tangents for H and H' using $\mathcal{O}(1)$ time:

Lemma 10.10. *Given an intersection H of circles and the convex polygon Q that is induced by the vertices of H, then each tangent l of Q can intersect at most two arcs of H.*

Proof. The proof is illustrated in Figure 10.6. We denote the edges of H by $e_1, \ldots, e_n$ and the edges of Q by $e'_1, \ldots, e'_n$ in such a way that e_i and e'_i connect the same vertices of H (Note that H and Q have the very same vertices). Since three consecutive vertices of H cannot be collinear, it is sufficient to consider the two cases whether l coincides with an edges e'_i of Q or not.

First let l not coincide with an edge of Q: Since l is a tangent for Q it still touches Q in a vertex v (see Figure 10.6). Let e'_i and e'_{i+1} be the two edges that are incident with v. Then e'_i and e'_{i+1} form a wedge which can be extended such that due to the convexity of Q it contains all edges of Q. We also conclude based on the convexity of H that all edges of H apart from e_i and e_{i+1} are contained in the same wedge. Thus, the tangent l can only intersect e_i and e_{i+1}.

If l coincides with an edge e'_i of Q it can only intersect e_i and all other edges lie to one side of l because of the convexity of H and Q. □

Given a tangent t between Q and Q' ending at vertices $v \in Q$ and $v' \in Q'$, according to the previous lemma we only have to consider the edges incident to v and v' in order to obtain a tangent t' for H and H' of the same type: We just compute the tangents of those arcs as depicted in Figure 10.7).

On that account we can compute the inner tangents for H and H' in $\mathcal{O}(\log(n_1) \cdot \log(n_2))$ time and the outer tangents in $\mathcal{O}(\log(n_1 + n_2))$ time:

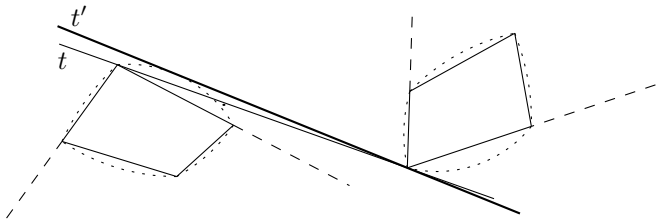


Figure 10.7: Illustration of an inner tangents between two intersections. The tangent t' is obtained by means of the tangent t.

Lemma 10.11. *Given two intersections H and H' based on circles $C_1, \ldots, C_n$ having a overlapping bound of k, then the inner and outer tangents between H and H' can be computed in $\mathcal{O}(\log^2 k)$ time.*

Assuming that we already have given the sequence $\mathcal{H} = (H_1, \ldots, H_m)$ we then can apply the approach of Section 6.2 in order to obtain a polygonal chain P'.

Lemma 10.12. *Given a sequence $\mathcal{H} = (H_1, \ldots, H_m)$ of consecutively disjoint intersections of the circles $C_1, \ldots, C_n$ having an overlapping bound of k, then we can find in $\mathcal{O}(n \cdot \log^2 k)$ time a polygonal chain P' such that*

1. *P' satisfies Property 6.1.*
2. *P' covers $\mathcal{H}$ with respect to its order.*
3. *P' has a minimum number of segments and inflection points regarding all polygonal chains satisfying the first two properties.*

Proof. The three properties can be directly concluded from the definitions and lemmas of the approach of Section 6.2. It remains to argue for the running time. To that end assume that we pre-calculate the tangents between the intersections in $\mathcal{H}$: For each pair (H_i, H_{i+1}) of intersections we need $\mathcal{O}(\log^2 k)$ time (Lemma 10.11). As we have to compute those tangents for all consecutive pairs we need $\mathcal{O}(m \cdot \log^2 k)$ time in total for all tangents. Since $n \geq m$ we derive $\mathcal{O}(n \cdot \log^2 k)$.

According to Theorem 6.4 we only need $\mathcal{O}(m) = \mathcal{O}(n)$ time for the rest of the procedure, that is, for computing P' based on the pre-computed tangents. □

So far we have assumed that we already have given the intersections $\mathcal{H}$, but normally we have to compute them based on a given polygonal chain P with connecting circles $C_1, \ldots, C_n$:

Theorem 10.3. *Given a polygonal chain P with not necessarily disjoint connecting circles $C_1, \ldots, C_n$ with overlapping bound k then we can find in $\mathcal{O}(k \cdot n)$ ($\mathcal{O}(k^2 \cdot n)$) time using the procedure `minIntersectionSequence` (`maxIntersectionSequence`) a sequence $\mathcal{H} = (H_1, \ldots, H_m)$ of consecutively disjoint intersections and a polygonal chain P' such that*

1. *P' satisfies Property 6.1.*
2. *P' covers $C_1, \ldots, C_n$ with respect to their order.*
3. *P' has a minimum number of segments and inflection points regarding all polygonal chains satisfying Property 6.1 and covering $\mathcal{H}$.*

Proof. The three properties directly follow from Lemma 10.12, so that we mainly have to argue for the running time. Using `minIntersectionSequence` we can compute $\mathcal{H}$ in $\mathcal{O}(n \cdot k)$ time according to Lemma 10.3. Using `maxIntersectionSequence` we can compute $\mathcal{H}$ in $\mathcal{O}(n \cdot k^2)$ time according to Theorem 10.1. And based on Lemma 10.12 we then can compute P' based on $\mathcal{H}$ in $\mathcal{O}(n \cdot k)$ time. □

We then can translate P' into a polyarc K using the approach of Section 6.1 in $\mathcal{O}(n)$ time (Theorem 6.1).

Due to this theorem we still can compute P' in $\mathcal{O}(n)$ time if we assume that k is constant for the considered polygonal chains P, which is a reasonable assumption regarding the considered use cases, as we already have argued for.

10.2 A Solution Based on Gates

In this section we describe how one can cover a polygonal chain P of size n by a polyarc K using the approach as described in Section 9.3 for the case that the connecting circles $C_1, \ldots, C_n$ of the corresponding corridor $\mathcal{C}$ may overlap.

To that end we first change the definition of feasible constellations: We do not require anymore that the gates must not intersect. On that account we also obtain feasible constellations $(g_1, g_2, L_2, R_2), \ldots, (g_{n-1}, g_n, L_n, R_n)$ based on $C_1, \ldots, C_n$ if those circles overlap (see Figure 10.8a). Figure 10.8a also illustrates the problem that arises: For two consecutive circles that overlap we can only find arcs that start or end wrongly at both gates. One possible solution would be to restrict the gates further in order to open a gap between both gates where the arcs can go through (see Figure 10.8b). But it is not obvious how to choose those restrictions in order to obtain optimal solutions.

The second possible solution, which we prefer, is to abdicate the requirement that an arc must start at the front of a gate and end at the back of the gate such that there are no intersections with that gate. On that account we change Definition 7.7 striking out the second requirement:

Definition 10.2. *Given two gates g_1 and g_2 with extensions e_1 and e_2, left obstacles $L = (l_1, ..., l_m)$ and right obstacles $R = (r_1, ..., r_m)$. Then an arc $A = (p_1, p_2, p_3)$ is called* valid *with respect to (g_1, g_2, L, R) if*

(1) p_1 lies on g_1 and p_3 lies on g_2 and

(2) ~~A does not start at g_1 wrongly and A does not end at g_2 wrongly and~~

(3) all obstacles of L lie to the left of A and

(4) all obstacles of R lie to the right of A and

(5) if two obstacles $o_i, o_j \in L \cup R$ with $i < j$ touch A, then the touching point of o_i lies before the touching point of o_j on A and

(6) if two obstacles $r_i \in R$ and $l_i \in L$ (of the same index) touch A, then either r_i touches A before l_i does or l_i touches A before r_i does.

If we use this definition we call the gates *permeable*. Note that this requirement has not been introduced for technical but only for aesthetic reasons in order to obtain a chain that does not intersect itself at the gates. Figure 10.8c illustrates this issue based on an

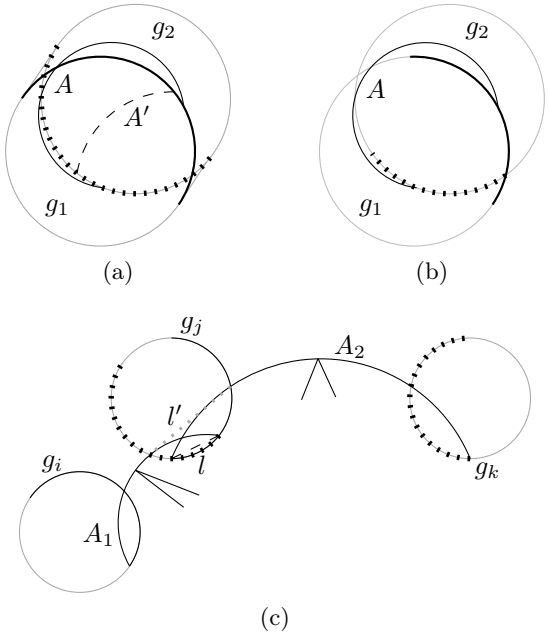


Figure 10.8: Illustration of permeable gates.

example consisting of three gates that are not necessarily consecutive. Applying the approach of Section 9.3 we normally obtain the chain that consists of A_1, l (black dashed line) and A_2 in that particular order. Since we introduce l in order to bridge the gap between A_1 and A_2, we also can choose another line that does not result in a crossing. For example, the line l' that is induced by the first intersection points of A_1 and A_2 with g_j lends itself for bridging the gap between A_1 and A_2.

10.3 Circles of Different Radii

Up to now we have omitted the question whether the connecting circles $C_1, \ldots, C_n$ of a give polygonal chain P of size n must have the same radius. From the practical point of view it can also be reasonable to allow different radii:

For example, imagine that one also want to model the behavior of zooming while following the trajectory (represented by P). In that case different radii can be used to adapt the polyarc K covering P to different zoom levels.

Further, if the circles model the permitted deviation of K regarding P it also can makes sense to allow different radii even if we not consider different zoom levels: If P leads through both a countryside and dense populated areas one probably wants to allow a greater deviation on the country side than in the dense populated area, because for the latter ones one want to have a focused view on P.

From the technical point of view, we can easily allow circles of different radii, if we also require that they do not overlap consecutively. Then we can apply both the line stabbing approach (Section 6.2) and the approach based on gates (Section 8.2) making only few adjustments:

1. For the line stabbing approach we have to choose an appropriate variant of the proce-

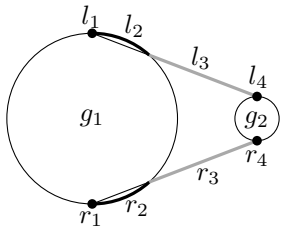


Figure 10.9: Corridor segment of two connecting circles of different radii. The obstacles of that segment are depicted boldly.

dure `coverCirclesWithLine`. As presented in 4.2 in [GHMS91] different variants are presented that can be used.

2. For the approach based on gates we only need to change some details of the corridor: For a corridor segment we have to introduce additional obstacles in order to model the borders correctly (see Figure 10.9). The gates can still be the connecting circles, because we do not require that the gates are equal.

We let the question remain open for future work how to handle connecting circles of different radii that are not consecutively disjoint. As circles can then contain other circles especially the approach of gates cannot adapted easily.

11. Conclusion

In this part we extensively discussed how to approximate a polygonal chain by a polyarc. To that end we started with a simple approach just replacing kinks of a polygonal chain by circular arcs and then in search of more sophisticated approaches we headed for two different directions:

The first direction is characterized by an immediate step transforming the given polygonal chain P' into another polygonal chain P' such that P' simplifies P. Then, we used one of the simple approaches to translate P' into a polyarc.

For the second direction we omitted an intermediate step and directly translated the given polygonal chain into a polyarc. In the following paragraph we summarize the results we have achieved for both main approaches.

Approach with Intermediate Step: In order to translate the given polygonal chain P into a simplified polygonal chain P' we used methods based on line stabbing in order to compute maximum long line segments covering P such that P' is assembled by those line segments. More in detail the idea is that the line segments stab circles that enclose the vertices of P.

As we want to resolve wiggly lines of P', we optimized P' for inflection points and number of segments. In [GHMS91] the simplification of P into P' is already solved, but they only proved that P' is of minimum length, but they did not prove that P' also consists of a minimum number of inflection points. In order to fill that gap, we introduced an alternative algorithm that persuades by its simplicity and by its extensibility. While the approach described in [GHMS91] is based on one line stabbing method, we can plug-in different stabbing methods as black-boxes such that the algorithm can be adapted to specific problems. Further, we proved that the algorithm yields a polygonal chain P' of minimum length and with minimum number of inflection points.

Later, in Chapter 10 we extended the approach to the case that we first want to summarize the given polygonal chain before we apply the line stabbing method. The idea is that when circles enclosing the vertices overlap then they are collapsed to their common intersection, such that after that step only consecutively disjoint intersections exist. For computing those intersections we presented two approaches:

The first approach considers the case that one wants to find as few intersections as possible in order to abstract the given polygonal chain P significantly. We gave a greedy algorithm that computes those intersections in $\mathcal{O}(k \cdot n)$ time, where n refers to the size of P and k denotes the overlapping bound of P.

The second approach focuses on the opposite case: We presented a greedy algorithm that computes a maximum number of consecutively disjoint intersection of those connecting circles in $\mathcal{O}(k \cdot n^2)$ time. Although the formulation of that algorithm is simple, the prove for its correctness is elaborate.

For computing the intersection n circles iteratively, it is likely that one can find a faster algorithm than we used so far. At the moment we make use of the naive variant requiring $\mathcal{O}(n^2)$ time. Even though there already is an algorithm computing the intersection of n circle within $\mathcal{O}(n \cdot \log n)$ time based on divide and conquer, for our approach it is crucial for a good performance to have an iterative variant at disposal. On page 91 we shortly sketch how

such a algorithm could look like.

Further research can also be done for the transformation from P' into K. So far we use only a simple approach for translating a kink into a circular arc. For aesthetic reasons it could be interesting to have further ways to translate P' into K at disposal.

Approach without Intermediate Step: We introduced an approach for gaining a polyarc K directly from a given polygonal chain P. To that end we generalized circle shooting: Up to now for circle shooting it has been assumed that one wants to compute for one given point the reachable points on a given line segment by means of circular arcs. The approach that we presented also provides the possibility to gain for a given line segment l all reachable points on another given line segment l' by means of circular arcs.

Based on the idea that each arc reaching l' from l must touch a certain order of obstacles we could show that there are five main cases to be considered. Those cases can then be used for gaining an algorithm that runs in $\mathcal{O}(n^2)$ time. Further, we also proved that l does not necessarily need to be a line segment but it also can be a circular arc.

Based on that approach different variants could be obtained for covering a polygonal chain P by a polyarc K. All of the variants have in common that they can solve generalization of the problem as presented in [DRS08]. The variants are:

1. Covering P with a not necessarily smooth polyarc K such that K consists of a minimum number of circular arcs ($\mathcal{O}(n^4)$ time).

2. Covering P with a not necessarily smooth polyarc K such that K consists of at most twice as many circular arcs as the optimal solution has ($\mathcal{O}(n^2)$ time).

3. Covering P wit a smooth polyarc K such that K consists of at most three times as many circular arcs as the optimal solution has ($\mathcal{O}(n^2)$ time).

We also considered special cases for which the connecting circles have different radii or overlap. In both case we motivated how to solve these problems.

Further research could be done for circle shooting by answering the question whether one can find a faster algorithm for processing circle shooting regarding line segments. To that end one could try to make use of Lemma 8.6 more efficiently.

For covering P by a polyarc one could also try to adapt the approach using biarcs as described in [DRS08] to the approach we are following: Is it possible for biarcs to formulate a similar lemma as Lemma 8.6?

Part II

Consistent Labeling Based on Trajectories

12. Introduction

We change our focus from polyarcs to maps, labels and trajectories: As already presented in Chapter 1 we consider the use case of a navigation system which offers to the observer a bird's eye view on the map: It presents a section of a requested route and moves along a trajectory that is based on that route as if we have a look through the lens of a camera moving over the map along that trajectory and adjusting its orientation to the direction of the trajectory. Hence, we call the currently visible area of the map the current *viewport* of the map.

Apart from containing the route, the map is also enriched with a set of points of interest that are described by means of rectangular labels that are closely placed to the corresponding points. In order to sustain the readability of the visible labels, all of these are horizontally aligned regarding the current viewport so that they begin to rotate when the viewport rotates (see Figure 12.1).

As the labels can overlap while rotating the viewport, we would like to have an optimal approach that describes how those labels can be switched off and on in order to avoid overlapping labels and in order to maximize the overall visibility of the presented labels. Consequently, we want to solve a special type of the dynamic map labeling problem ([BDY06]).

In this part of the thesis we analyze that particular labeling problem, which we call the trajectory based labeling problem (TBLP), in more detail assuming that the trajectory is given in form of a polyarc. We do not care about how that polyarc has been created, but we just use it. (For more information about creating polyarcs see the previous part of this thesis.)

We have chosen polyarcs to be representatives of trajectories, because they are assembled by a finite sequence of circular arcs, which can be easily described by simple geometric tools. We will extensively make use of that property in order to discretizes the labeling problem.

We have chosen the following structure for analyzing the labeling problem that we have only described roughly so far:

Chapter 13 – Related Work: We briefly describe the work that has already been done in the field of labeling maps. Similar to the equivalent of the first part we describe some work in more detail that is closely related to the problem we consider.

Chapter 14 – Model and Problem Definition: While so far we have explained the problem only informally, we present in that chapter a formal model that basically consists of maps, labels, trajectories and viewports. To that end we adapt the models as presented in

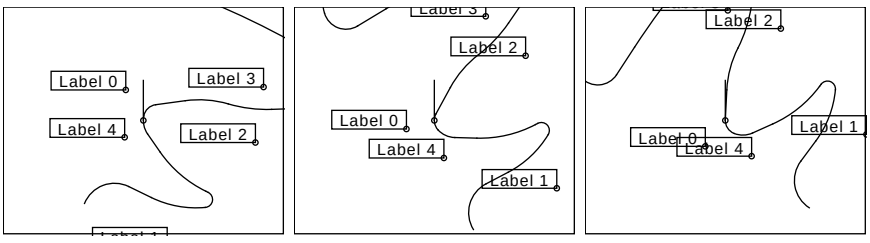


Figure 12.1: Viewport moving along a trajectory.

[BDY06] and [GNR11] and describe the terminology that arises when introducing viewports and trajectories. Finally, based on the introduced model we formally define the labeling problem that we want to consider.

Chapter 15 – Visibility and Conflicts: In that chapter we focus on the question how one can compute the visible labels within the currently chosen viewport. Further, we describe how conflicts between labels can be described and computed, such that those conflicts can be used for further analysis.

Chapter 16 – Complexity: In that chapter we answer theoretical questions about the complexity of the trajectory based labeling problem and show that the problem is $\mathcal{NP}$-complete. Further we introduce a conflict graph that can be obtained from an instance of trajectory based labeling problem and discuss its complexity.

Chapter 17 – Algorithmic Approaches: After proving the $\mathcal{NP}$-completeness of TBLP in the previous chapter, we model different variants of TPLB using integer linear programming in order obtain optimal solutions. Further, we introduce some simple heuristics that can be used as fast alternative for the presented ILPs.

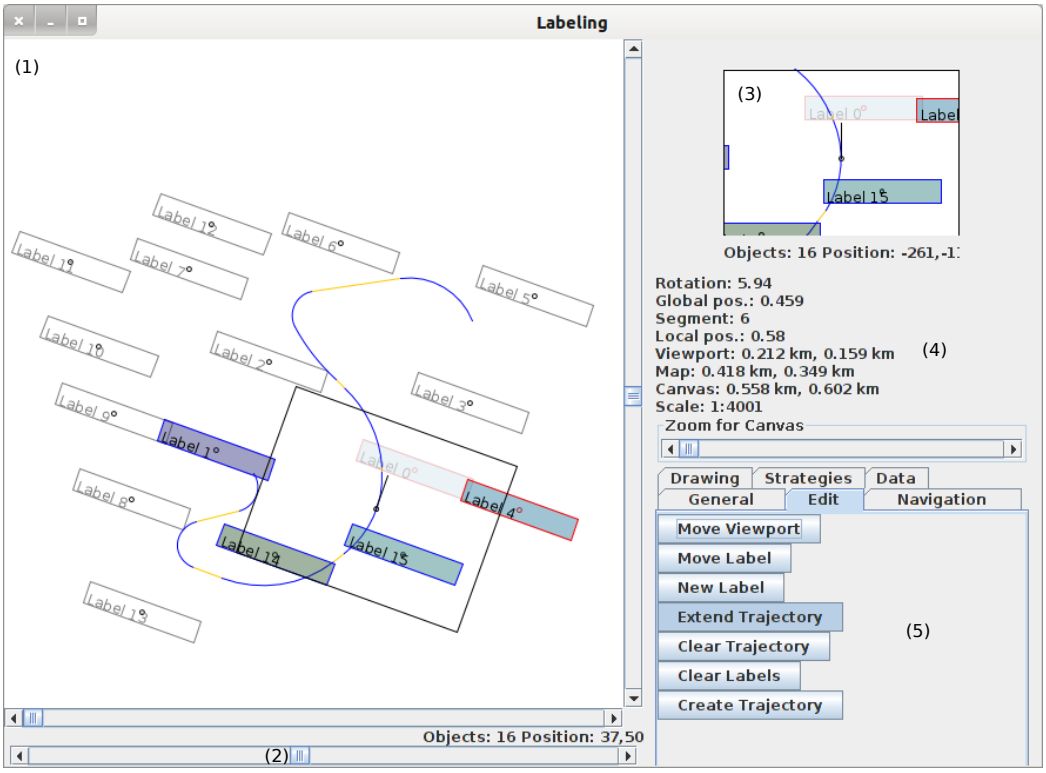
Chapter 18 – Experimental Evaluation: As we have implemented parts of the work, we present in that chapter the results of an experimental evaluation that we have applied on that implementation.

Chapter 19 – Conclusion: In the last chapter we summarize the results we have achieved and give a short outlook what next steps can be done for solving the labeling problem.

Apart from the theoretical results, we also have implemented basic parts that we present in this part of the work. In order to give the reader some information about that implementation, we present information boxes from time to time as it can be found on page 115:

These boxes contain details about the implementation such that the remaining thesis can be read without considering those information boxes. However, they contain useful information for readers who want to implement the presented work on their own. We use those boxes to discuss the design decisions we have made and show alternatives.

Implementation Details 12.1. The following figure shows a screenshot of the program we have implemented for this thesis:



- (1): An exemplary set of labels and a trajectory are presented within that frame of the application: The black rectangle illustrates the viewport at a certain position of the trajectory, while the colored labels are visible within the viewport. The gray label that overlaps with Label 4 is switched of in order to resolve the corresponding conflict.
- (2): By means of this bar the viewport can be moved along the trajectory such that the viewport and labels are adjusted to the direction of the trajectory.
- (3): This section of the application illustrates the currently selected viewport of the map such that it corresponds to the black rectangle as presented in frame (1).
- (4): Shows some general information about the current setting, as the number of segments of the trajectory and the actual dimension of the presented area.
- (5): This panel offers possibilities to interact with the application: For example the user can extend the trajectory or add labels. Further, it is possible to apply different strategies in order to compute an optimal solution for the labeling problem.

The main features of the tool are:

- Interactive creation of trajectories and label settings: The tool can be used as editor for trajectories and labels.
- Implementation of several ILPs and heuristics for handling the labeling problem.
- Support for real world data: It provides interfaces for converting OpenStreetMap[a] data into label data that can be loaded.
- Randomized creation of trajectories based on street maps: Shortest paths on street maps are used to gain randomized trajectories that can be used for tests.
- Evaluation: The tool offers an automatic mode with which evaluations can be applied.

[a] www.openstreetmap.org

13. Related Work

In this thesis we consider the *label number maximization problem* for rectangular labels: Given a set $L = \{l_1, \ldots, l_n\}$ of rectangular labels, the question is how to find a maximum cardinality subset S of L such that all labels of S can be drawn axis-aligned on a plane without overlap.

In many variants of that problem it is further assumed that a set of anchor points $P = \{p_1, \ldots p_n\}$ on a map is given for which a placement of the labels $L = \{l_1, \ldots, l_n\}$ should be found such that each label $l_i \in L$ lies close to its anchor point p_i. In [vKSW99, FW91] six different models are suggested how a label $l_i \in L$ can be placed with respect to its anchor point p_i (see Figure 13.1):

One Position: One fixed corner of l_i must coincide with p_i.

Two Positions: One of two fixed corners of l_i must coincide with p_i.

Four Positions: One of the four corners of l_i must coincide with p_i.

One Slider: A point of one fixed edge of l_i must coincide with p_i.

Two Sliders: A point of one of two fixed edges of l_i must coincide with p_i.

Four Sliders: A point of one of the four edges of l_i must coincide with p_i.

Since the first three cases assume that there is only a finite set of points on l_i where the anchor of l_i can lie, they are often called *fixed-position models*. The three remaining cases are often called *slider models*.

For all six cases it has been shown that the label number maximization problem is $\mathcal{NP}$-hard [FW91, MS91]. For a discussion about the complexity of those cases see [KM03]. For example the one-position model is a reformulation of the following problem that has been proven to be $\mathcal{NP}$-hard in [CC09]:

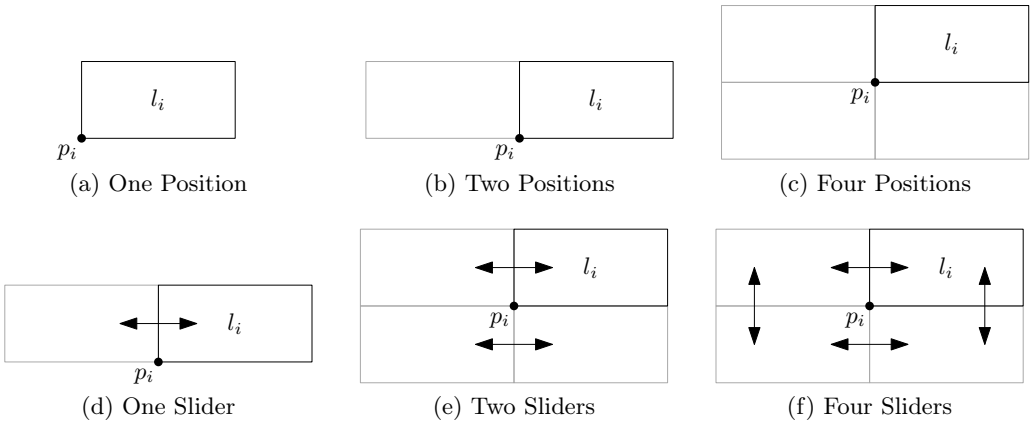


Figure 13.1: Placement of a label. Inspired by Figure 1 in [KM03].

Problem 13.1 (Maximum Independent Set of Rectangles (MISR))**.** *Given a set $\mathcal{R}$ of n axis-aligned rectangles in the plane, find a maximum cardinality subset S of $\mathcal{R}$ such that the rectangles in S do not overlap.*

Apart from the different cases of the placement of a label regarding its anchor, the labeling problem is divided in to *static labeling* and *dynamic labeling*:

The static labeling of a map is characterized by the property that one assumes that the labels, their position and their alignment do not change after the placement of the labels has been computed once. For that problem a typical use case are maps that are printed on paper.

Dynamic labeling considers a *dynamic map* for which the labeling can be changed over time. Been et al. [BDY06] suggest that a *dynamic map* is characterized *"by support for continuous zoom (change of scale) and continuous pan (change of region of interest)"*. Further research also comprises that the map may rotate [GNR11]. Thus, one can only see a section of the map and this section may change regarding scale, pan and rotation, which corresponds to the problem we want to consider.

In order to obtain an overview of dynamic labeling, the work presented in [BDY06] is a recommendable starting point. The authors introduce a general framework that subsumes many variants of dynamic labeling. Further, they describe important properties such that the labeling can be called *consistent*. To that end they call the location, size and orientation of a label its *placement* and the decision whether a label is shown or not the *selection* of a label. Then they require for a dynamic labeling that all of the following four statements are true:

Desiderata 13.1.

1. *"Except for sliding in or out of the view area, labels should not vanish when zooming in or appear when zooming out."*
2. *"As long as a label is visible, its position and size should change continuously under the pan and zoom operations."*
3. *"Except for sliding in or out of the view area, labels should not vanish or appear during panning."*
4. *"The placement and selection of any label is a function of the current map state (scale and view area)."*

While the first and third statement ensure that the labels do not pop up and flicker arbitrarily, the second requirement guarantees that the labels do jump. The last requirement enforces that the placement and the selection of a label is history independent.

Special cases of the dynamic map labeling problem are considered in [BNPW10, GNR11]: Been et al. consider in [BNPW10] continuous zoom of dynamic maps, whereat the size of a labels remains the same. The main idea is that each label possesses exactly one range of the possible zooming scale for which it is selected. For the time of that range they call the label *active*. Then the question is how those ranges can be chosen for all labels such that active labels do not overlap each other and the overall activity of the labels is maximized. They proved that this problem is $\mathcal{NP}$-complete for 1d- and 2d-maps. For the former one they describe a optimal solution using dynamic programming. Further, for both variants they introduce constant-factor approximations.

Gemsa et al. took a similar view on rotating maps in [GNR11]: They consider labels that rotate around fixed anchors such that each label may be selected for exactly one angle range.

Analogously to [BNPW10] they call a label *active* for that particular range. Consequently, the same question arises how choose those ranges for all labels such that active labels do not overlap each other and the overall activity of the labels is maximized. In order to answer the question they first introduce geometric tools for computing the angle intervals for which two labels overlap. Afterwards, they show that the corresponding decision problem is $\mathcal{NP}$-hard and consider therefore approximations of that problem: They introduce an $1/4$-approximation for that problem and generalize their approach to an efficient polynomial-time approximation scheme. As we do not consider zoom within this thesis, but only kinds of rotation we basically use [GNR11] as foundation for our work.

So far we have mainly presented theoretical work, but great effort has also been done on the practical side: For example in [PPH99, PGP03] Petzold et al. discuss a heuristic approach that works for dynamic maps supporting zooming and scrolling. In order to compute the labeling of huge data sets in realtime, they divide the approach in two phases. First they pre-compute a conflict graph for all labels containing information about possible conflicts between labels. This step may take some time, because it is applied only once. The second phase is determined for interactive requests and is therefore much faster than the first phase. For certain pan and zoom of the map they can guarantee that the labels do not overlap, but for continuous changes as described before they do not enforce the requirements as defined in Desiderata 13.1.

Another approach for computing the labeling of huge dynamic maps in realtime is presented in [LSC08]: Similar to the previous approach, the presented method is divided into a phase for pre-processing and a phase for requests. For computing overlaps between labels efficiently, labels are modeled by grouped particles. Then the remaining computations are done based on those particles such that requests can be processed in realtime. For each requested alignment and scale of a map it is guaranteed that there are no overlaps of labels. However, similar to the previous one, this approach does not enforce the requirements as defined in Desiderata 13.1.

14. Model and Problem Definition

In this chapter we describe in more detail how we model the labeling problem considered in this part of the thesis. In the following we introduce terminology which we will use later on. We first want to motivate that terminology roughly in order to give the reader an idea where we are heading for and then in separate paragraphs we describe this terminology in more detail.

Since we consider the labeling of a map, we obviously need concepts like *maps* and *labels*. We imagine a map to be a plane on which we can place labels at different locations describing *points of interest*. To that end we assume that labels are rectangles that are placed closely to the corresponding points they belong to using the one-position model as described in Chapter 13.

As we want to consider *dynamic* maps we have a dynamic view on the map, that is, we can only see a partial section of the map that can changes over time. We call that particular view, which we normally describe by a rectangle, the current *viewport* of the map. Further, we do not want the viewport to change arbitrarily, but determined by a pre-defined *trajectory* on the map, which is in our case a smooth polyarc.

So far we have introduced all concepts that we get as input for the labeling problem: A map M, a set $P = \{p_1, \ldots, p_n\}$ of points of interests, a set $L = \{l_1, \ldots, l_n\}$ that is related to P and a rectangle R describing the viewport with trajectory $T = (A_1, \ldots, A_m)$.

We require that the viewport is placed on the trajectory such that the center of the viewport is located at a point p on the trajectory. Apart from the location of the center we also want the viewport to have the same direction as the trajectory has at p. Consequently, if we move the viewport along the trajectory the viewport begins to rotate (see Figure 12.1).

Moving the viewport over the trajectory effects that labels appear and disappear in the viewport or more precisely they intersect the viewport. For a certain position on the trajectory we call all labels *visible* which intersect the viewport and the others *invisible*.

In order to preserve the readability of the labels we require that they remain horizontally aligned regarding the viewport. On that account it can happen that labels begin to overlap (see right most drawing of Figure 12.1) while the map is rotating. We call those phases of overlapping *conflicts* and the beginning and the end of those conflicts *conflict events*. We will resolve those conflicts by setting one of the involved labels *inactive* while the other remains *active*. We do not specify in detail how activity and inactivity of labels are represented, but normally we assume that the inactive label is faded out in a certain kind in order to guarantee the readability of the active label.

Then on an informal level the question to be answered is how labels can be set active and inactive such that all conflicts are resolved and still an optimal number of labels can be seen, which corresponds to the selection problem of [BDY06]. Obviously, we will also have to specify what *optimal* means.

It is helpful to imagine the viewport to be a rectangle that is moved over the given map along a pre-defined trajectory such that one has a general bird's eye view of the whole constellation (see Figure 14.1). Consequently, moving the viewport along the trajectory induces a sequence of rotations of the viewport. Since the labels are aligned with the axes of the viewport this sequence also implies a sequence of rotations of the labels.

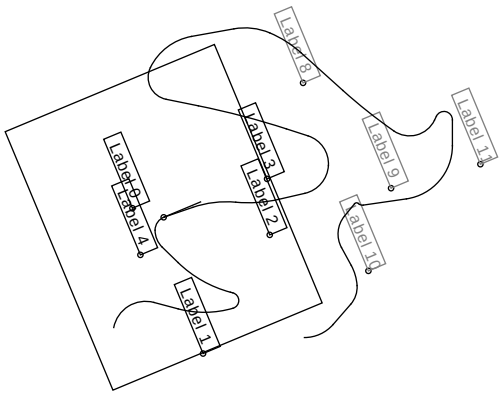


Figure 14.1: Bird's eye view on the map and the viewport, which is moved over the map.

After describing the problem informally we now want to define the terminology precisely.

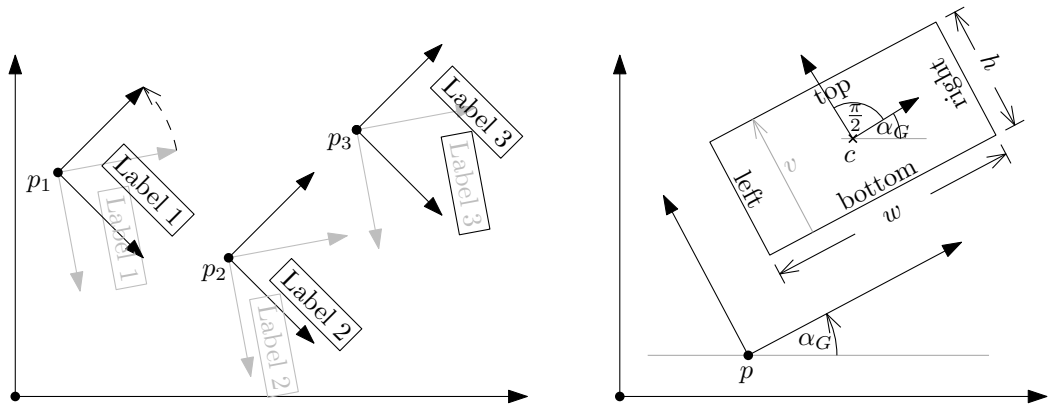
Map: We define the map M to be the plane $\mathbb{R}^2$. On the geometrical level we assume that we can draw shapes like line segments, arcs and rectangles on M. Thus, M is a kind of canvas. In order to describe the locations of those shapes, we assume that a Cartesian coordinate system underlies M which we denote by $\mathcal{CS}_M$. In accordance with computer graphics we call $\mathcal{CS}_M$ also the *world coordinate system*. Hence, when we refer to a point p on M we mean that its coordinates are related to $\mathcal{CS}_M$.

Points of Interest: We assume that apart from M we are also given a finite set $P = \{p_1, \ldots, p_n\}$ of points of interest. Those points are placed on M regarding $\mathcal{CS}_M$ such that all of them are pairwise disjoint.

In order to be independent of the world coordinate system, we introduce for each point $p \in M$ an own Cartesian coordinate system $\mathcal{CS}_p$. Hence, for a point p on M and a point $q \in \mathbb{R}^2$ we also write $q_{\mathcal{CS}_p}$ if we want to express that the coordinates of q are given in terms of $\mathcal{CS}_p$ and analogously $q_{\mathcal{CS}_M}$ if we talk about q regarding $\mathcal{CS}_M$. Normally, we assume that the affine transformation is done implicitly, so that we do not mention it if not necessary.

As we can define points in different coordinate systems we have to explain how those systems are related to each other: When we *draw* shapes specified in coordinates of $\mathcal{CS}_p$ on M we firstly apply an affine transformation such that the origin of $\mathcal{CS}_p$ coincides with p. In other words one can imagine that one draws the axes of the coordinate system $\mathcal{CS}_p$ on M, such that the intersection of both axes of $\mathcal{CS}_p$ coincides with p (see Figure 14.2a). Consequently, one degree of freedom remains, namely the rotation α of $\mathcal{CS}_p$ which is defined by the angle between the horizontal line through p (regarding $\mathcal{CS}_M$) and the x-axis of $\mathcal{CS}_p$ (see Figure 14.2b). According to custom we assume that α is measured using a counterclockwise orientation. From now on we also expect every angle to be normalized, that is, it lies between 0 and 2π. Since later on we want all labels to have the same rotation, we assume that α is the same for all points $p \in P$. Anticipating, we can say that α is dictated by the rotation of the viewport.

Anchored Rectangles: Before we define labels and the viewport more formally we introduce the concept of *anchored rectangles* for which we will show that they generalize both labels



(a) Several labels rotating around their anchor.

(b) An anchored counterclockwise oriented rectangle $R = (c, w, h)$ rotated by α_G.

Figure 14.2: The concept of anchors.

and viewports. To that end we distinguish the *left*, *right*, *top* and *bottom edge* of a rectangle, where the left (top) and right (bottom) edges lie opposite of each other (see Figure 14.2b). The length of the left and right edge is called *height*, while the length of the top and bottom edge is called *width*.

Then we call a rectangle $R = (c, w, h)$ with center c, width w and height h *anchored at a point* p if R is aligned with the axis of $\mathcal{CS}_p$, such that the bottom and top edge of R are parallel to the x-axis of $\mathcal{CS}_p$. Further, we require that the top edge of R lies above the bottom edge, that is, its y-coordinate is greater than the y-coordinate of the bottom edge. We call p the *anchor* of R. We also write $R = (\overline{l}, \overline{r}, \overline{b}, \overline{t})$ when we define R by its left edge $\overline{l}$, its right edge $\overline{r}$, its bottom edge $\overline{b}$ and its top edge $\overline{t}$.

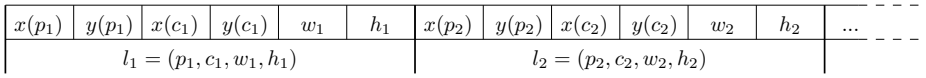
If we *draw* R on M, we always do this regarding $\mathcal{CS}_p$. Consequently, we first have to apply an affine transformation on R in order to obtain the coordinates for $\mathcal{CS}_M$. Since $\mathcal{CS}_p$ is rotated by the angle α_G, the rectangle R also has the rotation α_G regarding M.

It is easy to see considering M that if we increase α_G, the rectangle R begins to rotate on M around its anchor p in counterclockwise orientation.

Labels: According to common we assume that a label consists of some text or image describing a feature of the map. In order to abstract from the concrete shape of a label, we only consider the corresponding box. Thus, based on anchored rectangles we can easily introduce the concept of labels. For each point $p \in P$ we define its label as a rectangle $l = (c, w, h)$ which is anchored at p, that is, we draw l relatively to $\mathcal{CS}_p$ on M. We also write $l = (p, c, w, h)$ and do not require for c, w and h to be equal for all labels, but they can be different from each other. Further, in contrast to [GNR11] we do not require the anchor of l to lie within l or on its border, but it may be placed anywhere. However, for the application it is often usually the case that l is placed closely to p on M. We denote the set of all labels by L.

Note: As we require the center c of a label l to be fixed regarding the coordinate system $\mathcal{CS}_p$ of the anchor of l, we consider the one-position model as described in Chapter 13.

Implementation Details 14.1. In order to increase the performance, not for each label a individual object is created, but they are encoded in one array which is divided into blocks each representing one label:



This approach includes two benefits: First, instead of having overhead for each object, only the raw data is stored. Second, traversing the labels in a row the locality of the cache is utilized.
However, using Java for the implementation we prefer to have an object oriented view on labels. To that end we do not access the data directly but introduce a class `Label` wrapping individual blocks:

```
class Label{
  double [] dataOfAllLabels;
  int     index;

  public Label(double [] dataOfAllLabels, int labelNumber){
      this.index = labelNumber*size();
      this.data  = data;
  }

  public void next() {index += size();}
  public void previous() {index -= size();}
  public int  size() {return 6;}

  public double getAnchorX() {return data[index];}
  public double getAnchorY() {return data[index+1];}
  public double getCenterX() {return data[index+2];}
  public double getCenterY() {return data[index+3];}
  public double getWidth()   {return data[index+4];}
  public double getHeight()  {return data[index+5];}
}
```

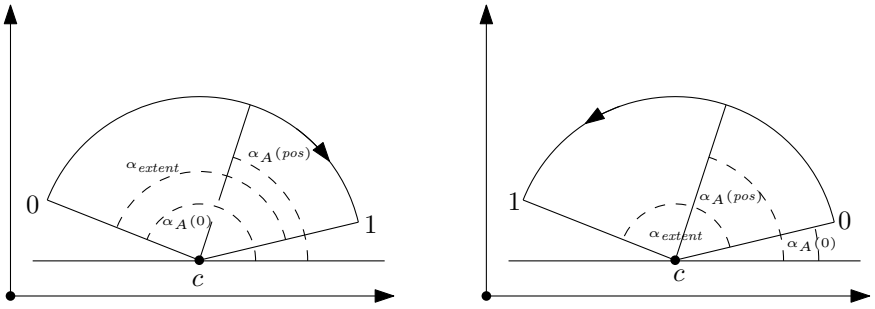
If we now iterate over the labels, we create exactly one instance of `Label` initializing it with the overall label data and the number of the label we want to consider. Using the methods `next` and `previous` we then can navigate through the data. On that account the class `Label` implements a kind of iterator.
Even though this data structure is static, it is sufficient for our purposes: We do not need to add or remove labels, but can assume that there is a pre-defined number of labels. The same applies for the following concepts as trajectories and conflicts. Thus, we also use the same approach for storing those ones.

Trajectory: Since a trajectory is a smooth polyarc, we first consider its building blocks, i.e., circular arcs. For a circular arc A we define the corresponding curve by a homonymous function $A\colon [0,1] \to M$. For A we call $[0,1]$ its *positions* and $A([0,1])$ its *locations* and denote the length of A by $|A|$.

The *rotation* $\alpha_A(pos)$ of an arc A at a position $pos \in [0,1]$ is defined by the angle that lies in between the horizontal line through the center c of A (regarding $\mathcal{CS}_M$) and the line through c and $A(pos)$ (see Figure 14.3). We define $\alpha_A(pos)$ regarding $\mathcal{CS}_M$ and not regarding $\mathcal{CS}_c$ because then we can connect arcs smoothly without having different rotations at the connection point of two arcs.

We call $\alpha_A(0)$ the *start angle* of A, $\alpha_A(1)$ the *end angle* of A and the angle range that is spanned by A the *extent* of A (Figure 14.3). Note that α_{extent} is independent of the orientation of A and therefore always positive. We can compute $\alpha_A(pos)$ depending on the orientation of A:

$$\alpha_A(pos) = \begin{cases} \alpha_A(0) - \alpha_{extent} \cdot pos & \text{if } A \text{ is clockwise oriented.} \\ \alpha_A(0) + \alpha_{extent} \cdot pos & \text{if } A \text{ is counterclockwise oriented.} \end{cases}$$



(a) A clockwise arc. (b) A counterclockwise arc

Figure 14.3: The concept of circular arcs.

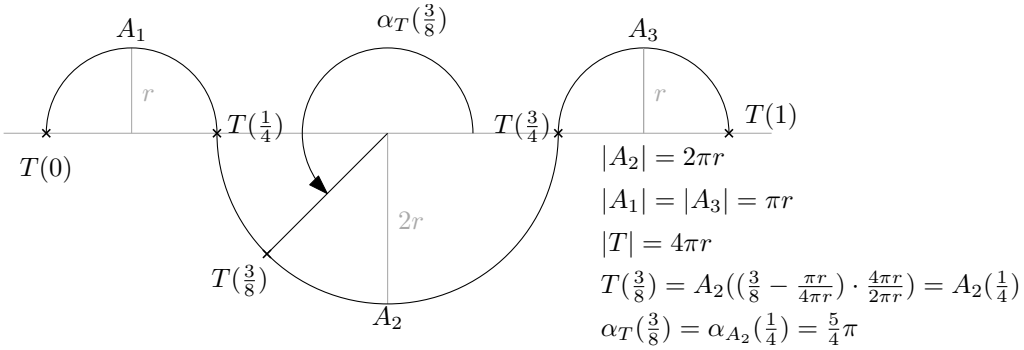


Figure 14.4: The concept of trajectories.

We still have to explain how positions and locations of arcs are related to each other. To that end we define for all positions $pos \in [0,1]$ of an arc A with center $c = (x_c, y_c)$ and radius r:

$$A(pos) = (x_c + \cos(\alpha_A(pos)) \cdot r, y_c + \sin(\alpha_A(pos)) \cdot r)$$

Based on those arcs we define a trajectory $T\colon [0,1] \to M$ to be a smooth curve on M that is assembled by m circular arcs $A_1, \ldots, A_m$ (see Figure 14.4). Analogously to arcs we call $[0,1]$ the *positions* of T and $T([0,1])$ the *locations* of T.

In order to distinguish positions of single arcs and of the given trajectory we also call those for arcs *local* positions and those for the trajectory *global* positions. The length of T is defined as $|T| = \sum_{i=1}^{m} |A_i|$.

Now we define the function $\lg_T\colon \{A_1, \ldots, A_m\} \times [0,1] \to [0,1]$ mapping local on global positions for an arc A_i of T as

$$\lg_T(A_i, pos) = \frac{\sum_{j=1}^{i-1} |A_j|}{|T|} + pos \cdot \frac{|A_i|}{|T|}$$

For $\lg_T(A_i, 0)$ we also write $\text{begin}_T(A_i)$ and for $\lg_T(A_i, 1)$ we also write $\text{end}_T(A_i)$. In order to explain the opposite direction we introduce a function $\text{arc}_T\colon [0,1] \to \{1, \ldots, m\}$ that maps each global position on an unique arc by its index:

$$\text{arc}_T(pos) = \begin{cases} 1, & 0 \leq pos \leq \frac{|A_1|}{|T|} \\ i, & 2 \leq i \leq m \text{ and } \frac{\sum_{j=1}^{i-1} |A_j|}{|T|} < pos \leq \frac{\sum_{j=1}^{i} |A_j|}{|T|} \end{cases}$$

Then we can introduce the function $\mathrm{gl}_T : [0,1] \to \{A_1, \ldots, A_m\} \times [0,1]$ mapping global positions on local positions:

$$\mathrm{gl}_T(pos) = (A_i, (pos - \frac{\sum_{j=1}^{i-1} |A_j|}{|T|}) \cdot \frac{|T|}{|A_i|}) \text{ with } i = \mathrm{arc}_T(pos)$$

Due to the definition of arc_T the definition gl_T is well-formed: The weld point of two consecutive arcs is mapped on the first of both arcs. Obviously, then it is true that for all $pos \in [0,1]$: $\mathrm{lg}_T(\mathrm{gl}_T(pos)) = pos$. Finally, we can formally define T for a position $pos \in [0,1]$ as:

$$T(pos) = A_i(pos') \text{ with } (A_i, pos') = \mathrm{gl}_T(pos)$$

The definition of $T(pos)$ implies that the global positions $[0,1]$ of T are spread over T in a linear way, that is, going from $T(0)$ to a location $T(pos)$ along the trajectory we cover the fraction $pos \cdot |T|$ of the length of T.

At last we define the *rotation* $\alpha_T(pos)$ of T at the global position pos as

$$\alpha_T(pos) = \alpha_{A_i}(pos') \text{ with } (A_i, pos') = \mathrm{gl}_T(pos).$$

Implementation Details 14.2. Assuming that the trajectory T is pre-defined, we can compute the global positions of the beginning and ending of the single arcs in advance gaining an array of intervals:

$$[\mathrm{begin}_T(A_1), \mathrm{end}_T(A_1)], (\mathrm{begin}_T(A_2), \mathrm{end}_T(A_2)], \ldots, (\mathrm{begin}_T(A_m), \mathrm{end}_T(A_m)]$$

Then, when we need to compute $arc_i(pos)$ for a global position $pos \in [0,1]$ we can do this in $\mathcal{O}(\log m)$ time using a simple binary search on those intervals.

Viewport: After defining the trajectory we now can explain the viewport $\mathcal{VP}$ in more detail, which, as already mentioned, represents the visible section of the map. Recall that for analysis we take a very general bird's eye view where we can see the map and the viewport moving along the trajectory such that its rotations corresponds to the rotation of T (see Figure 14.1). On that account we represent the viewport by means of a rectangle of corresponding size such that the center c of that rectangle coincides with a point of T. In particular we say that this rectangle has the position $pos \in [0,1]$ on T if $c = T(pos)$. We identify that rectangle with $\mathcal{VP}$ and denote it therefore also by $\mathcal{VP}$.

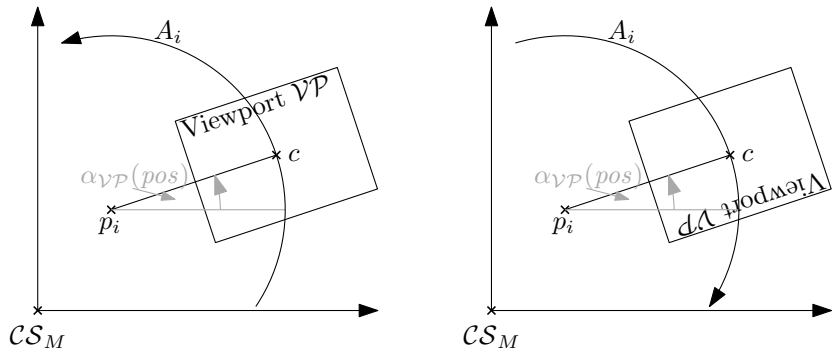
The rotation $\alpha_{\mathcal{VP}}(pos)$ of $\mathcal{VP}$ at the global position $pos \in [0,1]$ is defined as the rotation of the trajectory T at pos: $\alpha_{\mathcal{VP}}(pos) = \alpha_T(pos)$. Note that the rotation of $\mathcal{VP}$ does not distinguish the two cases whether the underlying arc A_i (with $i = \mathrm{arc}_T(pos)$) is clockwise or counterclockwise oriented (see Figure 14.5). Consequently, drawing the viewport on the map the orientation of A_i must be comprised separately.

Since the given trajectory T is composed of arcs $A_1, \ldots, A_m$, going along T we can split the motion of $\mathcal{VP}$ into m phases of rotation where each phase corresponds to exactly one arc:

$$[\mathrm{begin}_T(A_1), \mathrm{end}_T(A_1)], (\mathrm{begin}_T(A_2), \mathrm{end}_T(A_2)], \ldots, (\mathrm{begin}_T(A_m), \mathrm{end}_T(A_m)]$$

Thus, we can imagine that for each phase or rotation the rectangle $\mathcal{VP}$ is anchored to the center p_i of A_i and rotates around p_i when it moves along A_i (see Figure 14.5). On that account it is reasonable to consider the phases of rotation separately from each other and then to assemble the results canonically. For the arc A_i we denote the anchored rectangle representing $\mathcal{VP}$ by $R(\mathcal{VP}, A_i)$.

It is then easy to see that restricting $\mathcal{VP}$ to an arc A it has the same behavior as a label of same size and same anchor. We therefore often can analyze anchored rectangles instead of the viewport and the labels and we then can transfer the results on both the viewports and the labels.



(a) A clockwise arc A_i. (b) A counterclockwise arc A_i.

Figure 14.5: Orientation of the viewport $\mathcal{VP}$. The rotation $\alpha_{\mathcal{VP}}(pos)$ does not comprise the orientation of the viewport, but it must be considered separately: For both cases $\alpha_{\mathcal{VP}}(pos)$ is equal.

Visibility: Considering the movement of $\mathcal{VP}$ along T, there are certain events, where the given labels begin or end to intersect $\mathcal{VP}$, or in other words those labels begin or end to be (partly) visible in $\mathcal{VP}$. For a certain position $pos \in [0,1]$ we call all labels that intersect $\mathcal{VP}$ *visible* at *pos* and all other labels *invisible* at *pos*. More formally we describe the visibility of a label regarding $\mathcal{VP}$ and T by means of the function $\Phi_T^{\mathcal{VP}}$: $[0,1] \times L \to \{0,1\}$:

$$\Phi_T^{\mathcal{VP}}(pos, l) = \begin{cases} 1, & l \text{ intersects } \mathcal{VP} \text{ at position } pos \text{ on } T \\ 0, & \text{otherwise} \end{cases}$$

Thus, for each label l we define the visibility as:

Definition 14.1. *Given a label l and a viewport $\mathcal{VP}$ with trajectory T, then the sequence*

$$\Phi_T^{\mathcal{VP}}(l) = ([s_1, e_1], \ldots, [s_k, e_k])$$

is called the visibility *of l if*

1. *k is minimized and*
2. *$0 \leq s_1 < e_1 < s_2 < e_2 < \ldots < s_k < e_k \leq 1$ and*
3. *for all $pos \in [0,1]$ it is true that*

$$\Phi_T^{\mathcal{VP}}(pos, l) = 1 \text{ if and only if there exists an } i \text{ with } 1 \leq i \leq k \text{ such that } pos \in [s_i, e_i]$$

Note that this means in particular that the intervals contained in $\Phi_T^{\mathcal{VP}}(l)$ are disjoint. If for an arc A of T and a visibility range $v{=}[s,e] \in \Phi_T^{\mathcal{VP}}(l)$ of l it is true that

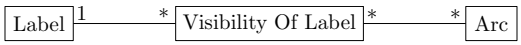
$$[\text{begin}_T(A), \text{end}_T(A)] \cap [s,e] \neq \emptyset \text{ if } A \text{ is the first arc of T and}$$
$$(\text{begin}_T(A), \text{end}_T(A)] \cap [s,e] \neq \emptyset \text{ otherwise}$$

we say that v and A *overlap* and otherwise they are *disjoint*.

If we want to have an isolated view on A, we conceptually assume that T only consists of A. In that case we also write $\Phi_A^{\mathcal{VP}}(l)$ for the sequence of visibilities of l. Obviously, then

local and global positions coincide. Later on in Chapter 15 we explain in more detail how we gain those intervals and how results for a single arc can be turned into corresponding results for a trajectory consisting of several arcs.

Implementation Details 14.3. Based on the following class diagram there are among other things two ways to store the visibilities of a label l:



1. Either each label l owns a container which stores those visibilities,
2. or one stores all visibilities in one huge array (as described in Implementation Details 14.1) and relates each arc A of T to the visibilities of l that overlap with A. Consequently, one visibility v of a label l may point to severals consecutive arcs.

On the first sight the first one seems to be reasonable, since conceptually moving along the trajectory each label owns visibilities, which also means that there is a one-to-many relation. But from the side of efficiency the second one is preferable:

1. If we draw the viewport and its content, we only want to draw labels which are at least visible. For the first variant we have to go through all labels in order to select those visible labels. Even, if we use concepts as quad-trees [dBvKOS97] in order to reduce the number of visited labels, we still consider labels which are not drawn. Applying the second variant we can restrict ourselves to the labels for which a visibility points to the arc the current position belongs to. From those ones we know that they are the only ones that can be visible at the current position. On that account we often can restrict the set of labels to be drawn significantly.
2. Further, assume a server-client scenario: The clients request from visible labels the server for different trajectories. For the first variant this means that we have to annotate the labels with information of different clients, which means additional overhead in order to keep those data structures consistent, so that different requests do not interfere each other. But if we use the second variant, the labels and their data structures are not effected, because we only read on those labels, so that the server can handle the data structures used for the trajectories, independently. Consequently, especially if we want to work on the map and its labels concurrently, the second variant is the better choice.

Conflicts: Since the rotation of labels corresponds to the rotation of the viewport $\mathcal{VP}$ it can happen that while moving $\mathcal{VP}$ along the given trajectory T two labels begin to intersect (see Figure 14.1). Since we want to avoid those intersections, we call them *conflicts*. There are two possibilities to describe conflicts:

1. We just consider the case that the labels are rotated simultaneously without taking the trajectory and the viewport into account: As mentioned in [GNR11] we can then describe those conflicts by means of intervals referring to angles that correspond to the conflict. We call this the *general description* of conflicts. Obviously, it is sufficient to consider the angle interval $[0, 2\pi]$ in order to describe those conflicts completely.

   For two labels l and l' we can describe all conflicts between them by a sequence of maximal disjoint angle intervals $c_1 = [\alpha_1, \beta_1], \ldots, c_k = [\alpha_k, \beta_k]$. We then define the

following three sets

$$\begin{aligned} C(l,l') =& \{[\alpha_1,\beta_1],\ldots,[\alpha_k,\beta_k] \mid \text{ all maximal long disjoint conflict intervals} \\ & \text{between } l \text{ and } l'\} \\ C(l) =& \bigcup_{l'\in L} \{([\alpha,\beta],l') \mid [\alpha,\beta] \in C(l,l')\} \\ C(L) =& \bigcup_{l\in L} \{\{[\alpha,\beta],l,l'\} \mid ([\alpha,\beta],l') \in C(l)\} \end{aligned}$$

We also often write $c = (l, l')$ for a conflict $c = \{[\alpha, \beta], l, l'\}$, if we are not interested in the interval itself, but only in the labels involved in c.

2. The other kind of description comprises the trajectory T and is therefore called *trajectory dependent description*: For two labels $l, l' \in L$ we only consider conflicts $c \in C(l, l')$ that are also visible within the viewport for a certain position *pos* on T. In particular this means that for *pos* the viewport intersects the intersection of l and l':

$$\Phi_T^{\mathcal{VP}}(pos, c) = \begin{cases} 1, & \text{the intersection of } l \text{ and } l' \text{ intersects } \mathcal{VP} \text{ at the position } pos \\ 0, & \text{otherwise} \end{cases}$$

Analogously to labels we can describe the visibility of a conflict c regarding a trajectory T and a viewport $\mathcal{VP}$ by a sequence of position intervals (Figure 14.6 shows that a conflict $c \in C(l, l')$ can have several visibilities):

Definition 14.2. *Given a conflict $c = (l, l')$ and a viewport $\mathcal{VP}$ with trajectory T, then the sequence*

$$\Phi_T^{\mathcal{VP}}(c) = ([s_1, e_1], \ldots, [s_k, e_k])$$

is called the visibility *of c if*

a) k is minimized and

b) $0 \leq s_1 < e_1 < s_2 < e_2 < \ldots < s_k < e_k \leq 1$ and

c) for all $pos \in [0, 1]$ it is true that

$$\Phi_T^{\mathcal{VP}}(pos, c) = 1 \text{ if and only if there exists an } i \text{ with } 1 \leq i \leq k \text{ such that } pos \in [s_i, e_i]$$

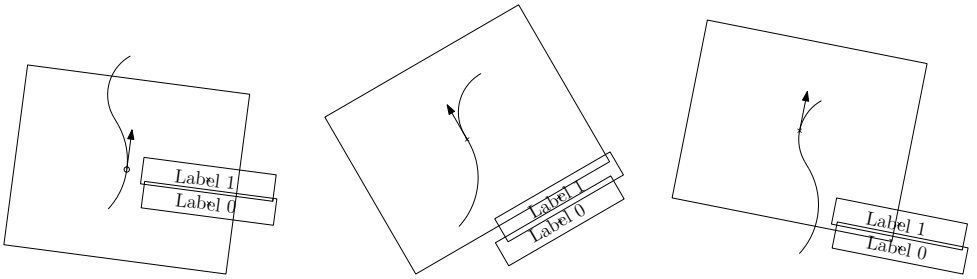
Again we have defined the sequence in such way that it contains only disjoint intervals describing all visibilities of c completely.

In Chapter 15 we explain in more detail in which way both descriptions are related to each other.

We *resolve* a conflict between two labels l_1 and l_2 applying the common approach that we set one of both labels *inactive* for a certain period of time while the other remains *active*. In particular we require that only visible labels can be active.

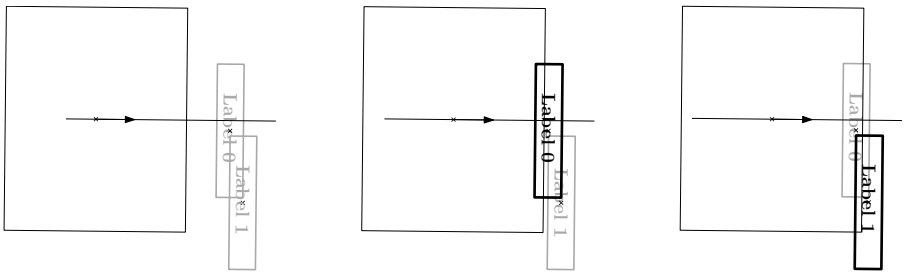
There are several ways to represent inactivity of a label l, for example one could stop drawing it for that period of time or, the approach we prefer, one draws the inactive label softly in the background.

Now, we explain which strategies we apply in order to handle conflicts based on the trajectory-dependent conflict description. To that end we introduce the notion of being a witness of setting a label active or inactive:



(a) (b) (c)

Figure 14.6: Visibility of conflicts. The same conflict can be visible several times. (The centers of the labels correspond with the anchors of the labels.)



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

Figure 14.7: Visibility of conflicts. A conflict may arise before it becomes visible (black label=active label, gray label=inactive label).

Definition 14.3. *Given a set L of labels and a viewport $\mathcal{VP}$ with trajectory T.*

1. *If we set a label $l \in L$ inactive at a position pos of T and at this position a conflict $c{=}(l, l') \in C(L)$ begins, we say that l'* is a witness for setting l inactive.

2. *If we set a label $l \in L$ active at a position pos of T and at this position a conflict $c{=}(l, l') \in C(L)$ ends, we say that l'* is a witness for setting l active.

Note that both definitions imply that l' is only a witness, when the beginning/ending of the visibility of c is also the beginning/ending of c itself. Figure 14.7 shows a case where c begins before it becomes visible. In that case we want to avoid the behavior as depicted: First Label 0 becomes visible and active, then when Label 1 becomes visible, the activity switches from Label 0 to Label 1, which seems to be distracting. We also want to omit the analogous behavior for disappearing labels. For that purpose we demand the following activity model for labels:

Definition 14.4. *Given a set L of labels and a viewport $\mathcal{VP}$ with trajectory T, then we distinguish three types of **A**ctivity **M**odels regarding $\mathcal{VP}$:*

AM1: *A label may only become active when it becomes visible in $\mathcal{VP}$ and it may only become inactive when it stops to be visible in $\mathcal{VP}$.*

AM2: *Same as AM1, but the label may also become inactive at any position pos, if at pos there is an active witness l' for setting l inactive.*

AM3: *Same as AM2, but the label may also become active at any position pos, if at pos there is an active witness l' for setting l active.*

Since AM1 and AM2 are special cases of AM3, we only illustrate the third case in Figure 14.9, page 134. For all three types we can introduce a function $\Psi_T^{\mathcal{VP}}\colon [0,1] \times L \to \{0,1\}$ which states whether a label l is active at a position $pos \in [0,1]$:

$$\Psi_T^{\mathcal{VP}}(pos, l) = \begin{cases} 1 & l \text{ is active at position } pos \text{ of } T \\ 0 & \text{otherwise} \end{cases}$$

In general, in order to avoid distracting effects as demanded in [BDY06], we require that a label l has **only one phase of activity for each of its visibilities**. Finally, we can express the optimization problem that we want to solve as follows:

Problem 14.1. *Given a map M with points of interest $P = \{p_1, \ldots, p_n\}$ and corresponding labels $L = \{l_1, \ldots, l_n\}$ and a viewport $\mathcal{VP}$ with trajectory $T = (A_1, \ldots, A_m)$.*

Then we want to find a concrete instance of $\Psi_T^{\mathcal{VP}}$ such that

1. *all conflicts of active labels are resolved regarding one of the three activity models AM1, AM2 or AM3 and*
2. *for each $l \in L$ and each visibility range $[s,e] \in \Phi_T^{\mathcal{VP}}(l)$ there is at most one phase $[pos_1, pos_2] \subseteq [s,e]$ of activity and*
3. *for each label $l \in L$ and all positions $pos \in [0,1]$ it is required that $\Psi_T^{\mathcal{VP}}(pos, l_i) = 1$ implies $\Phi_T^{\mathcal{VP}}(pos, l) = 1$ and*
4. *the sum*

$$\sum_{i=1}^{n} \int_0^1 \Psi_T^{\mathcal{VP}}(pos, l_i) dpos$$

is maximum.

Since due to the integral over $\Psi_T^{\mathcal{VP}}$ this problem definition is not very practical for actually computing it, we divide the visibility of a label l into single disjoint segments represented by position intervals. The idea is that single conflict events of the conflicts of l induce the transition between two intervals:

Definition 14.5. *Given a set L of labels, a viewport $\mathcal{VP}$ with trajectory T and a label $l \in L$ with its conflicts $C(l)$, then*

$$S_T^{\mathcal{VP}}(l) := (pos_1, \ldots, pos_k)$$

is called the events *of T regarding the label l if*

$$S_T^{\mathcal{VP}}(l) = \{0,1\} \cup \{s, e \mid [s,e] \in \Phi_T^{\mathcal{VP}}(l)\} \cup \{s, e \mid [s,e] \in \Phi_T^{\mathcal{VP}}(c) \text{ and } c \in C(l)\}$$

In particular we call each interval $s_i = [pos_i, pos_{i+1}]$ (with $1 \leq i < k$) a segment *of T regarding l and abbreviate this by $s_i \in S_T^{\mathcal{VP}}(l)$ or by $[pos_i, pos_{i+1}] \in S_T^{\mathcal{VP}}(l)$. We therefore call $S_T^{\mathcal{VP}}(l)$ also the segments of T regarding the label l.*

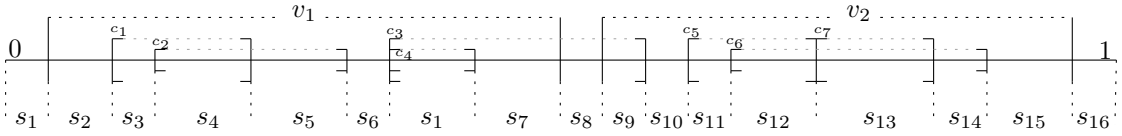


Figure 14.8: Illustration of segments of a trajectory regarding a label.

Figure 14.8 illustrates the definition: Every time a conflict or visibility of l begins or ends, a new segment begins.

Before we reformulate the problem, we extend the activity $\Psi_T^{\mathcal{VP}}$ of a label to intervals:

$$\Psi([pos_1, pos_2], l) = \begin{cases} 1 & \forall pos \in [pos_1, pos_2]\colon \Psi_T^{\mathcal{VP}}(pos, l) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Then we can reformulate Problem 14.1 to:

Problem 14.2 (Trajectory Based Labeling Problem (TBLP))**.**
Given a map M with points of interest $P = \{p_1, \ldots, p_n\}$ and corresponding labels $L = \{l_1, \ldots, l_n\}$ and a viewport $\mathcal{VP}$ with trajectory $T = (A_1, \ldots, A_m)$. Further, let $w : L \times [0,1] \times [0,1] \to \mathbb{R}$ be a weight function.

Then we want to find a concrete instance of $\Psi_T^{\mathcal{VP}}$ such that

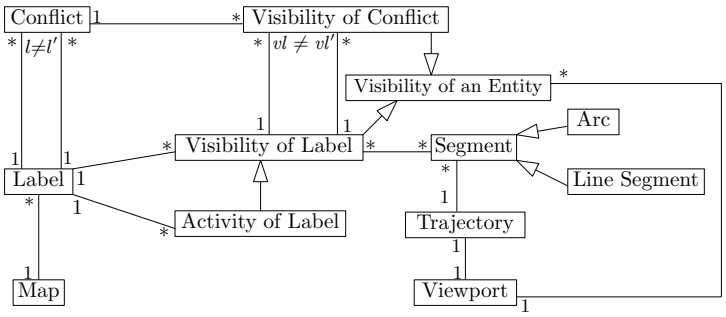
1. *all conflicts of active labels are resolved regarding one of the three activity models AM1, AM2 or AM3 and*
2. *for each $l \in L$ and each visibility $[s, e] \in \Phi_T^{\mathcal{VP}}(l)$ there is at most one phase $[pos_1, pos_2] \subseteq [s, e]$ of activity and*
3. *for each label $l \in L$ and all positions $pos \in [0,1]$ it is required that $\Psi_T^{\mathcal{VP}}(pos, l_i) = 1$ implies $\Phi_T^{\mathcal{VP}}(pos, l) = 1$ and*
4. *the sum*

$$\sum_{l \in L} \sum_{[s,e] \in S_T^{\mathcal{VP}}(l)} \Psi_T^{\mathcal{VP}}([s, e], l) \cdot w(l, s, e)$$

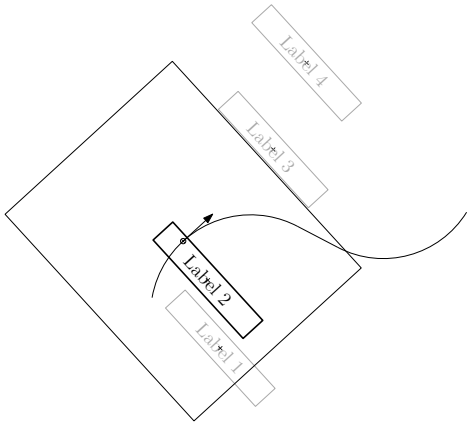
is maximal.

If we set the weight function to $w(l, s, e) = e - s$ for all labels $l \in L$ and all segments $[s, e] \in S_T^{\mathcal{VP}}(l)$, it is easy to see that both problem definitions are identical. Further, for all three activity models for labels the segments in $S_T^{\mathcal{VP}}(l)$ are the smallest units that can be independently active.

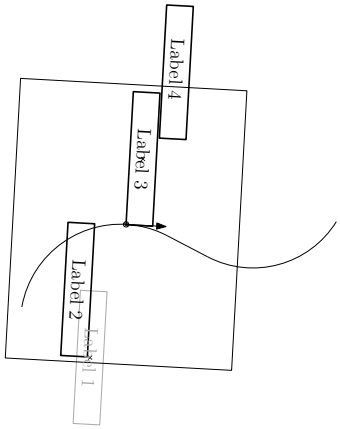
Implementation Details 14.4. The following diagram shows a sketch of a class diagram how the different concepts can be related to each other:



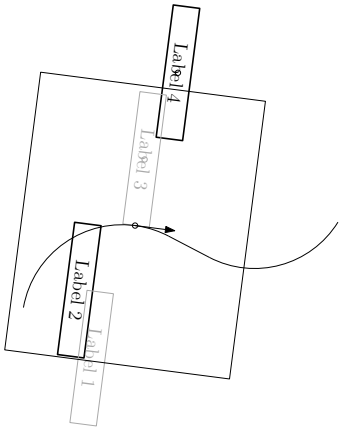
Explanation of the class diagram: A *map* consists of several *labels* which can be in *conflict* independently from the *viewport* and its *trajectory*. If there is a conflict for a label l then there is a second label l' which is the antagonist of l. A conflict always consists between two labels only.
Normally, there are some labels which are visible within the viewport $\mathcal{VP}$ while it moves along the trajectory T. This *visibility of a label* is a special type of the *visibility of an entity*. One also can imagine other entities as conflicts that can be visible. In all cases the visibility is closely connected to one viewport. Furthermore a label can be active for several intervals, then it owns severals *activities* which is a special case of being visible.
If two labels l and l' have one conflict c in common, this conflict can be visible at different positions. This *visibility of a conflict* is both related to the conflict c and to the visibility vl of l and the visibility vl' of l'. Since it is a kind of visibility, it is a special type of a visibility of an entity.
The viewport is related to exactly one trajectory and vise versa. Each trajectory consists of a sequence of segments which either are arcs or line segments. Even though a line segment is the extreme case of an arc with infinite radius, it is reasonable to distinguish both types of segments, because we only have finite data types at our disposal. Thus, we also introduce some threshold for the planeness of an arc such that falling short of this threshold for an specific arc we interpret it as a line segment.



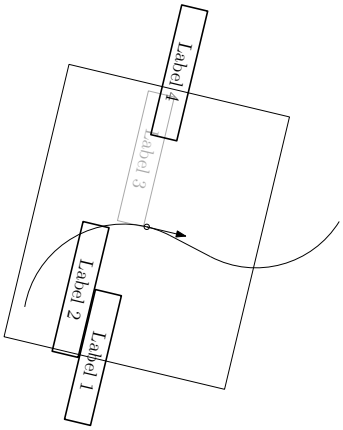
(a) 1. Position: Only Label 2 is active, because Label 3 and Label 4 are not visible within the viewport yet and Label 1 awaits the end of the impending conflict with Label 2 before it becomes active.



(b) 2. Position: Label 1 and Label 2 are in conflict, so that Label 1 is still inactive. In the meanwhile both Label 3 and Label 4 have entered the viewport and became active.



(c) 3. Position: Label 1 and Label 2 are still in conflict and Label 3 and Label 4 are also in conflict. Label 3 has lost the conflict and is therefore inactive since the conflict has begun.



(d) 4. Position: Label 1 is active because the conflict with Label 2 ended a bit earlier. Label 3 and Label 4 are still in conflict, but even after the conflict is over, Label 3 will remain inactive, because it has already been active.

Figure 14.9: Resolving conflicts. Illustrates the resolving of conflicts by means of type AM3 (black label=active label, gray label=inactive label). The anchors of the labels correspond to the centers of the labels.

15. Visibility and Conflicts

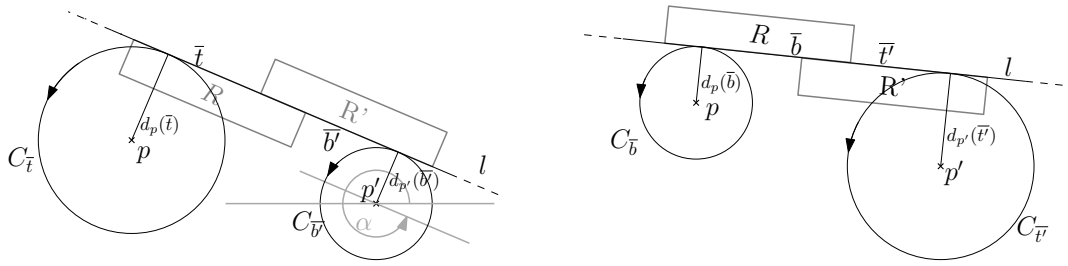
In this chapter we want to explain in more detail how a closed description of conflicts between anchored rectangles (label-label-conflict, label-viewport-visibility) can be obtained, that is, we explain how the intersection of two rotating anchored rectangles can be described precisely. In [GNR11] it is already shown that between two anchored rectangles there can exist at most eight conflicts, but the description is restricted to rectangles where the anchor lies within the rectangle. In the first section we describe how anchored rectangles can also be handled for those whose anchor lies outside of their borders. In the second section we then explain how this approach can be used in order to obtain a general description of conflicts between labels. Afterwards the third section also comprises the viewport and its trajectory describing how visibilities of labels and conflicts can be obtained. Finally, in the last section we discuss, how the approach can be extended by considering the area of labels that are visible.

15.1 Anchored Rectangles and Their Conflicts

In this section we consider two anchored rectangles $R = (\bar{l}, \bar{r}, \bar{b}, \bar{t})$ and $R' = (\bar{l}', \bar{r}', \bar{b}', \bar{t}')$ as defined in Chapter 14, with anchors p and p', and describe how the conflict events between both rectangles can be obtained.

To that end we assume that both rectangles have the same rotation α regarding the world coordinate system and that when we *rotate* them, we do this in a *counterclockwise orientation*, which allows us to distinguish the *beginning* and the *end* of a conflict. Note that the restriction to counterclockwise rotations is without loss of generality: For clockwise oriented rotations, we just have to switch the notions of *beginning* and *end*. As already stated in Chapter 14 a conflict event is an angle α for which a conflict between R and R' either begins or ends (see Figure 15.1). On that account we head for a sequence of angles within $[0, 2\pi]$ describing all conflicts of R and R' within this interval.

More in detail: Since a conflict between two anchored rectangles R and R' is defined based on the non-empty intersection of both rectangles, a rotation angle α can only be a conflict



(a) Begin of a conflict.

(b) End of a conflict.

Figure 15.1: Conflict events. Assuming that R and R' rotate in counterclockwise orientation first $\bar{t}$ and $\overline{b'}$ mark the beginning of the conflict and then $\bar{b}$ and $\overline{t'}$ mark the end of the conflict.

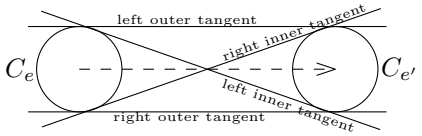


Figure 15.2: Tangents of C_e and $C_{e'}$

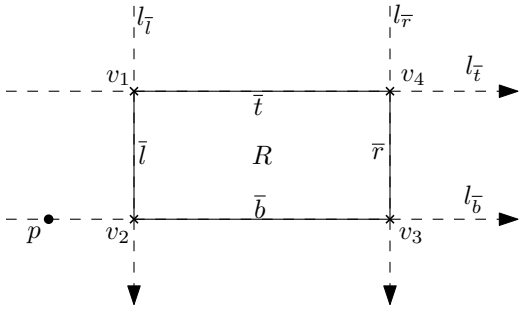


Figure 15.3: Anchor position of an anchored rectangle.

event if there is an edge e of R and an edge e' of R' such that both edges lie on a common line l (see Figure 15.1). In particular e and e' must be opposite sided of R and R', that means, that if for example e is the top of R, then e' must be the bottom of R'.

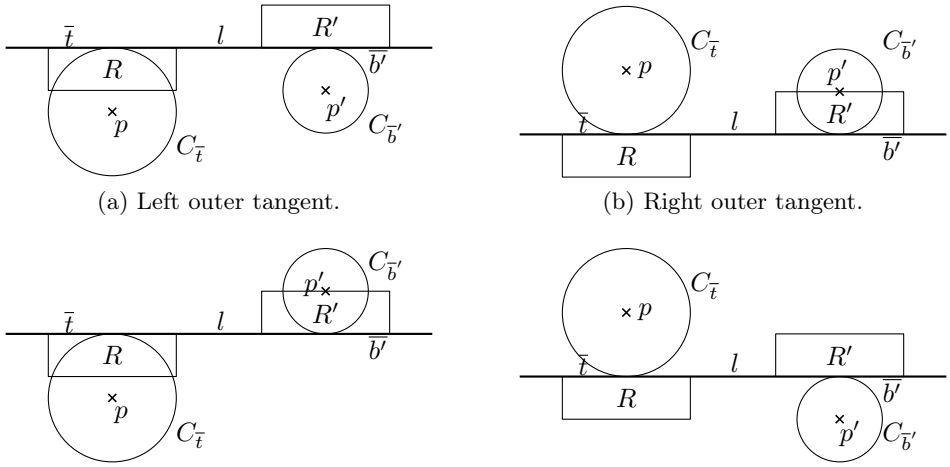
We call that line l the *tangent* of the conflict event α: The term is motivated by the fact that l is a tangent between the circles C_e and $C_{e'}$ around p and p' of radius $d_p(e)$ and $d_{p'}(e')$, respectively, where $d_p(e)$ describes the distance between the line through the line segment e and the point p (in Figure 15.1 $C_{\bar{t}}$ and $C_{\bar{b}}$ are depicted). We can imagine the existence of l as a necessary requirement for a conflict event. Analyzing l we therefore do not necessarily require that R and R' intersect. Later on we explain a further requirement by which we check whether R and R' indeed intersect at that particular event.

For the further reasoning it is crucial to distinguish four types of tangents. Due to simple geometry l can be either an *inner tangent* or an *outer tangent* (see Figure 15.2). Moreover, going from C_e to $C_{e'}$ the line l is a *left tangent* if C_e lies to the right of l and a *right tangent* if C_e lies to the left of l. Consequently, we have to distinguish the four cases that l is a left/right inner/outer tangent.

We now explain how we have to interpret those four cases regarding anchored rectangles. To that end we first consider a single anchored rectangle (R, p) with its four edges $(\bar{l}, \bar{r}, \bar{b}, \bar{t})$ and its four corners v_1, v_2, v_3 and v_4 as depicted in Figure 15.3. If we extend the edges of R to oriented lines $l_{\bar{t}} = (v_1, v_4)$, $l_{\bar{l}} = (v_1, v_2)$, $l_{\bar{b}} = (v_2, v_3)$ and $l_{\bar{r}} = (v_3, v_4)$, we can distinguish whether p lies to the left or to the right of an edge of R. For instance in Figure 15.3 the anchor p lies to the right of $\bar{l}$, $\bar{r}$ and $\bar{t}$. For the special case that p lies on $l_{\bar{t}}$, $l_{\bar{l}}$, $l_{\bar{b}}$ or $l_{\bar{r}}$, we define that p lies to the side of that line that corresponds to the interior of R. On that account in Figure 15.3 the anchor p lies to the left of $\bar{b}$. As R is defined relatively to $\mathcal{CS}_p$ this so defined relation between p and the edges of R does not change when we transform R to $\mathcal{CS}_M$.

After we have introduced this terminology, we return the two anchored rectangles R and R' and describe when we have to consider each of the four cases. Due to symmetry we only consider the case that the top edge $\bar{t}$ of R and the bottom edge $\bar{b}'$ of R' lie on a common line l. For the other cases (bottom/top, left/right, right/left) we can argue analogously.

Let $C_{\bar{t}}$ be the tangential circle for the edge $\bar{t}$ and let $C_{\bar{b}'}$ be the tangential circle for the



(a) Left outer tangent.

(b) Right outer tangent.

(c) Left inner tangent.

(d) Right inner tangent.

Figure 15.4: Tangents for different anchor positions.

Position of p	Position of p'	Type of l
p lies to the **right** of e.	p' lies to the **right** of e'.	**left outer** tangent of C_e and $C'_{e'}$
p lies to the **right** of e.	p' lies to the **left** of e'.	**left inner** tangent of C_e and $C'_{e'}$
p lies to the **left** of e.	p' lies to the **right** of e'.	**right inner** tangent of C_e and $C'_{e'}$
p lies to the **left** of e.	p' lies to the **left** of e'.	**right outer** tangent of C_e and $C'_{e'}$

Table 15.1: Given two anchored rectangles (R, p) and (R', p') the table lists all possible tangents of two equally aligned edges e and e' of R and R'.

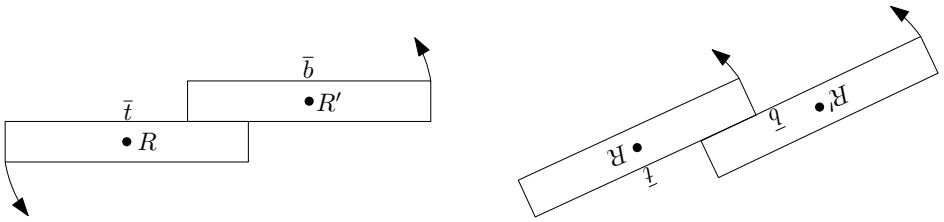
edge $\overline{b}'$ (see Figure 15.4). If the anchor p lies to the right of $\bar{t}$ and the anchor p' lies to the right of $\overline{b}'$, l must be the left outer tangent of $C_{\bar{t}}$ and $C_{\overline{b}'}$, because both anchors lie on the same side of l, namely the right side (going from $C_{\bar{t}}$ to $C_{\overline{b}'}$). Analogously we can argue, that if p lies to the left of $\bar{t}$ and p' lies to the left of $\overline{b}'$, then l must be the right outer tangent of $C_{\bar{t}}$ and $C_{\overline{b}'}$.

If p lies to the right of $\bar{t}$ and p' lies to the left of $\overline{b}$, the same must apply for the line l. Thus, l is the left inner tangent of $C_{\bar{t}}$ and $C_{\overline{b}'}$. Analogously, l is the right inner tangent of $C_{\bar{t}}$ and $C_{\overline{b}'}$ if p lies to the left of $\bar{t}$ and p' lies to the right of $\overline{b}$.

For two edges e and e' of R and R' that are opposite sides of R and R' Table 15.1 summarizes the results in general.

In the next step we use l to gain the rotation angle α of possible conflict events between R and R'. To that end we describe conflict events relative to the edges of R: In the general case we can find for each edge of R two conflict events. One that describes the beginning of a conflict and one that describes the end of a conflict. For instance in Figure 15.5a the top edge $\bar{t}$ of R induces the beginning of a conflict if we assume that we rotate both rectangles in counterclockwise orientation around their anchors, while in Figure 15.5b the edge $\bar{t}$ induces the end of a conflict assuming the same orientation. Obviously, considering two edges $e \in R$ and $e' \in R'$ describing opposite sides of R and R', if e induces the beginning of a conflict for an angle α, then e' also does.

In the following we consider the conflicts induced by the edges of R in more detail. To that



(a) $\overline{t}$ induces the beginning of a conflict. (b) $\overline{t}$ induces the end of a conflict.

Figure 15.5: Inducing conflict events. The top edge $\overline{t}$ of R induces the beginning and the end of a conflict.

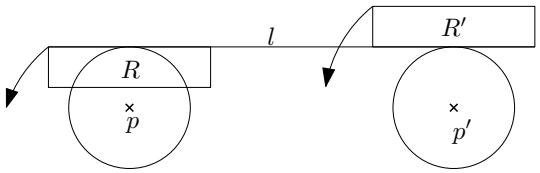


Figure 15.6: Lying on a common line is only a necessary condition for conflict events.

end we distinguish conflict events induced by horizontal and vertical edges. For the next part be aware of the fact that we only analyze a necessary condition for conflict events, namely that two edges inducing a conflict event must lie on the same line l at this conflict event. Figure 15.6 shows that this does not necessarily mean that a conflict begins or ends actually. Later on we explain in more detail how to use this condition in order to check for conflict events.

Top Edge: Assume that we simultaneously rotate R and R' in such a way that $\overline{t}$ and $\overline{b}'$ lie on a common line l. Due to Table 15.1 there are four main cases distinguishing the types of l. Since there are two angles for which $\overline{t}$ and $\overline{b}'$ lie on a common line (inducing a beginning and an end of a possible conflict), we divide each case into two sub-cases.

Since all four main-cases are very similar, we only discuss one case in more detail with its two sub-cases. For the remaining three cases we only present corresponding drawings.

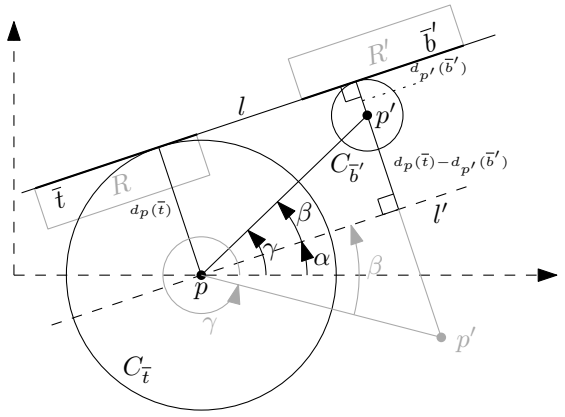
First we consider the main-case that p lies to the right of $\overline{t}$ and p' lies to the right of $\overline{b}'$. According to Table 15.1 the tangent l must be the left outer tangent of $C_{\overline{t}}$ and $C_{\overline{b}'}$ (see Figure 15.7a and Figure 15.7b).

We first consider the sub-case that the conflict event describes the possible beginning of a conflict, that is, if we continue rotating the labels in counterclockwise orientation the rectangles could begin to intersect (see Figure 15.7a: if the labels lay close enough, they would begin to intersect). Then l and the x-axis of the world coordinate system enclose the angle α which describes the rotation of both anchored rectangles. We cannot directly gain α, but we can consider two other angles which assemble α:

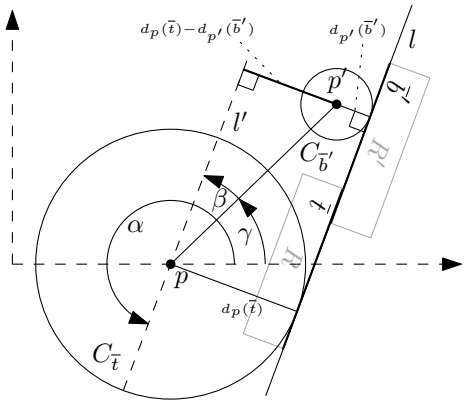
1. The slope of the line $\overline{pp'}$ that we can describe by the angle $\gamma = \arctan_2(y(p')-y(p), x(p')-x(p))$[1] forms a relation between the world coordinate system and both rectangles.

[1] We define $\arctan_2$ as

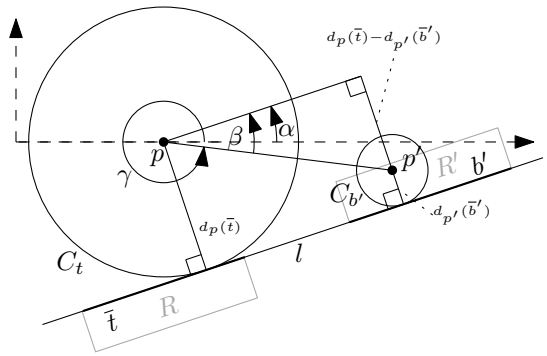
	$x > 0$	$y \geq 0, x < 0$	$y < 0, x < 0$	$y>0, x=0$	$y<0, x=0$	$y=0, x=0$
$\arctan(y,x)_2 =$	$\arctan(\frac{y}{x})$	$\arctan(\frac{y}{x}) + \pi$	$\arctan(\frac{y}{x}) - \pi$	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	–



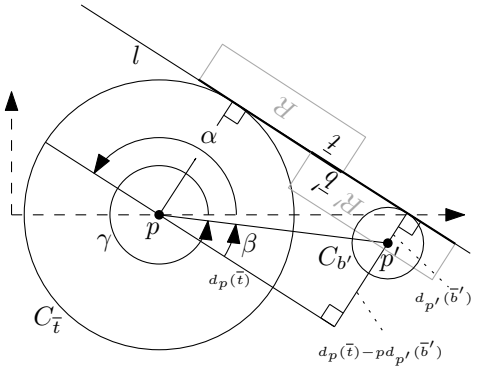
(a) **Begin** of a possible conflict: p lies to the **right** of $\bar{t}$, p' lies to the **right** of $\bar{b}'$.



(b) **End** of a possible conflict: p lies to the **right** of $\bar{t}$, p' lies to the **right** of $\bar{b}'$.



(c) **Begin** of a possible conflict: p lies to the **left** of $\bar{t}$, p' lies to the **left** of $\bar{b}'$.



(d) **End** of a possible conflict: p lies to the **left** of $\bar{t}$, p' lies to the **left** of $\bar{b}'$.

Figure 15.7: Considering outer tangents the figure shows the conflict events for the top edge of R and the bottom edge of R'.

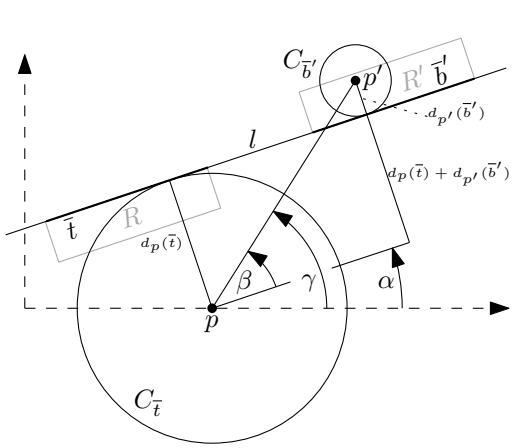
2. The angle $\beta = \arcsin(\frac{d_p(\bar{t})-d_{p'}(\bar{b}')}{\|p-p'\|_2})$ is the angle between the line segment $\overline{pp'}$ and the line l' that is parallel to l going through p.

Then we can easily derive $\alpha = \gamma - \beta$. Note that this also holds if $C_{\bar{b}'}$ is larger than $C_{\bar{t}}$. In that case the term $\arcsin(\frac{d_p(\bar{t})-d_{p'}(\bar{b}')}{\|p-p'\|_2})$ becomes negative (due to $\arcsin(-x) = -\arcsin(x)$), that is, $\alpha = \gamma + \arcsin(\frac{d_{p'}(\bar{b}')-d_p(\bar{t})}{\|p-p'\|_2})$ which corresponds to the gray drawings in Figure 15.7a. On that account we do not have to distinguish which of both circles is larger. For the following cases we can argue analogously so that we will not mention it again.

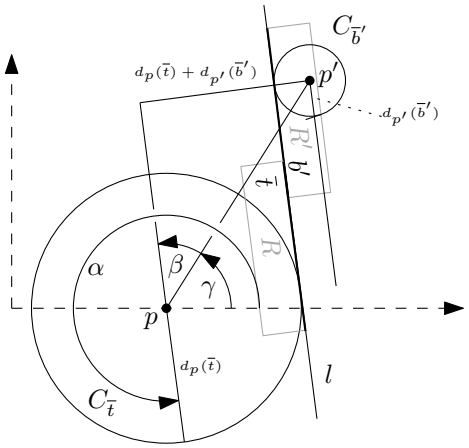
In the case that the conflict event describes the ending of a conflict we can proceed in the same way (see Figure 15.7b), but this time it is true that $\alpha = \gamma + \beta + \pi$.

We do not discuss the remaining cases: For the case that both anchors lie to the left of both edges see Figure 15.7c and Figure 15.7d. For the cases that l is an inner tangent see Figure 15.8: The main difference is that $\beta = \arcsin(\frac{d_p(\bar{t})+d_{p'}(\bar{b}')}{\|p-p'\|_2})$.

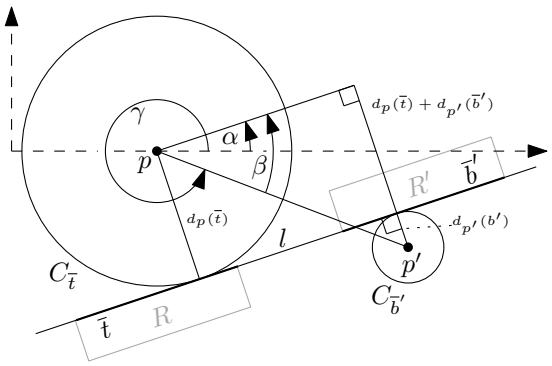
in order to circumvent special cases.



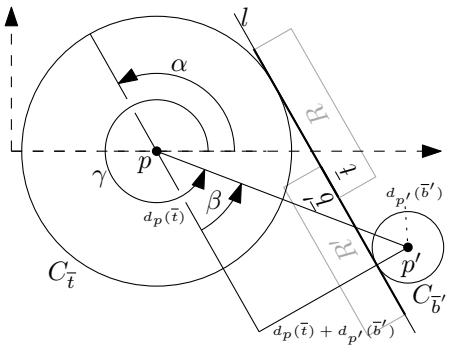
(a) **Begin** of a possible conflict: p lies to the **right** of $\overline{t}$, p' lies to the **left** of $\overline{b}'$.



(b) **End** of a possible conflict: p lies to the **right** of $\overline{t}$, p' lies to the **left** of $\overline{b}'$.



(c) **Begin** of a possible conflict: p lies to the **left** of $\overline{t}$, p' lies to the **right** of $\overline{b}'$.



(d) **End** of a possible conflict: p lies to the **left** of $\overline{t}$, p' lies to the **right** of $\overline{b}'$.

Figure 15.8: Considering inner tangents it shows the conflict events for the top edge of R and the bottom edge of R'.

Table 15.2 summarizes the results. Note that two resulting angles of a particular case do not belong to the same conflict, that is, the start angle belongs to the beginning of another conflict than the end angle belongs to. Later on when we have described the conflict events for all other edges, we also explain how those angles can be composed to a consistent sequence of conflict events.

Bottom Edge: Since possible conflict events induced by the top edge $\overline{t}$ of R corresponds to the conflict events of the bottom edge $\overline{b}'$ of R', we do not need to consider the conflict events induced by the bottom edge $\overline{b}$ of R explicitly. We can compute them switching the roles of R and R', that is, we compute the conflict events induced by the top edge $\overline{t}'$ of R' as described in the previous case. Those conflict events then are the very same as those induced by $\overline{b}$.

R	R'	Begin	End
p lies to the **right** of $\bar{t}$.	p' lies to the **right** of $\bar{b}'$.	$\alpha = \gamma - \beta_1$	$\alpha = \gamma + \beta_1 + \pi$
p lies to the **right** of $\bar{t}$.	p' lies to the **left** of $\bar{b}'$.	$\alpha = \gamma - \beta_2$	$\alpha = \gamma + \beta_2 + \pi$
p lies to the **left** of $\bar{t}$.	p' lies to the **right** of $\bar{b}'$.	$\alpha = \gamma + \beta_2$	$\alpha = \gamma - \beta_2 + \pi$
p lies to the **left** of $\bar{t}$.	p' lies to the **left** of $\bar{b}'$.	$\alpha = \gamma + \beta_1$	$\alpha = \gamma - \beta_1 + \pi$
α =conflict event, $\gamma = \arctan_2(y(p') - y(p), x(p') - x(p))$, $\beta_1 = \arcsin(\frac{d_p(\bar{t}) - d_{p'}(\bar{b}')}{\|p-p'\|_2})$,$\beta_2 = \arcsin(\frac{d_p(\bar{t}) + d_{p'}(\bar{b}')}{\|p-p'\|_2})$			

Table 15.2: Conflict Events of the Top Edge - For two anchored rectangles (R, p) and (R', p) the table shows the possible conflict events for the top edge $\bar{t}$ of R and the bottom edge $\bar{b}'$ of R'. It is assumed that R and R' are rotated in counterclockwise orientation.

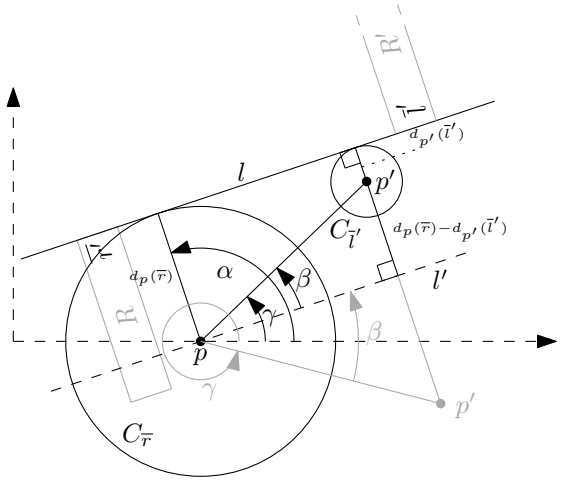
R	R'	Beginning	End
p lies to the **right** of $\bar{l}$.	p' lies to the **right** of $\bar{r}'$.	$\alpha = \gamma - \beta_1 + \frac{\pi}{2}$	$\alpha = \gamma + \beta_1 - \frac{\pi}{2}$
p lies to the **right** of $\bar{l}$.	p' lies to the **left** of $\bar{r}'$.	$\alpha = \gamma - \beta_2 + \frac{\pi}{2}$	$\alpha = \gamma + \beta_2 - \frac{\pi}{2}$
p lies to the **left** of $\bar{l}$.	p' lies to the **right** of $\bar{r}'$.	$\alpha = \gamma + \beta_2 + \frac{\pi}{2}$	$\alpha = \gamma - \beta_2 - \frac{\pi}{2}$
p lies to the **left** of $\bar{l}$.	p' lies to the **left** of $\bar{r}'$.	$\alpha = \gamma + \beta_1 + \frac{\pi}{2}$	$\alpha = \gamma - \beta_1 - \frac{\pi}{2}$
α =conflict event, $\gamma = \arctan_2(y(p') - y(p), x(p') - x(p))$, $\beta_1 = \arcsin(\frac{d_p(\bar{r}) - d_{p'}(\bar{l}')}{\|p-p'\|_2})$,$\beta_2 = \arcsin(\frac{d_p(\bar{r}) + d_{p'}(\bar{l}')}{\|p-p'\|_2})$			

Table 15.3: Conflict events of the right edge. For two anchored rectangles (R, p) and (R', p) the table shows the possible conflict events for the right edge $\bar{r}$ of R and the left edge $\bar{l}'$ of R'. It is assumed that R and R' are rotated in counterclockwise orientation.

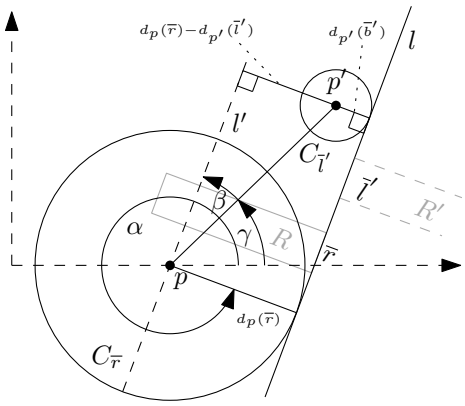
Right Edge: For the right edge $\bar{r}$ of R we again can reuse the results of the top edge: The only difference is that the rotation angle α of R and R' does not correspond with the rotation angle of l but has an offset of $\frac{\pi}{2}$ (the right and left edge of an anchored rectangle are vertical edges and have therefore a rotation of $\frac{\pi}{2}$ when the rectangle has a rotation of 0). In order to give the reader the possibility to verify those results we present in Figure 15.9 and Figure 15.10 the corresponding drawings, but do not explain them explicitly because they are very similar to those ones in Figure 15.7 and Figure 15.8. Table 15.3 shows the description of the possible conflict events assuming that the right edge of R and the left edge of R' induce the conflict.

Left Edge: Assume that the left edge $\bar{l}$ of R and the right edge $\bar{r}'$ of R' lie on a common line. Analogously, to the bottom edge we can argue that we do not need to consider the conflict events explicitly, but that we can reuse the results of the case for the right edge: We just switch roles between R and R', because we know that the conflict events induced by $\bar{r}'$ are the very same as the ones induced by $\bar{l}$.

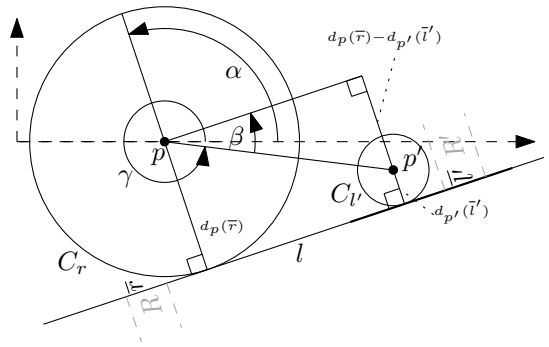
Checking for Real Conflicts: So far we only have discussed a necessary condition for conflict events, namely that the two edges inducing the conflict event must lie on a common line at this particular angle, but as depicted in Figure 15.6 it is not necessarily the beginning or end of a conflict. Thus, we still have to describe a criterion which helps us to testify conflicts. To that end we rotate both rectangles to the possible conflict event α we want to check and project the vertices of both rectangles lying on the common line l either on the x-axis or on



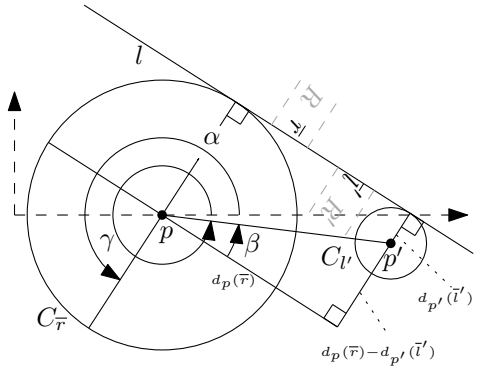
(a) **Begin** of a possible conflict: p lies to the **right** of $\overline{r}$, p' lies to the **right** of $\overline{l}'$.



(b) **End** of a possible conflict: p lies to the **right** of $\overline{r}$, p' lies to the **right** of $\overline{l}'$.



(c) **Begin** of a possible conflict: p lies to the **left** of $\overline{r}$, p' lies to the **left** of $\overline{l}'$.

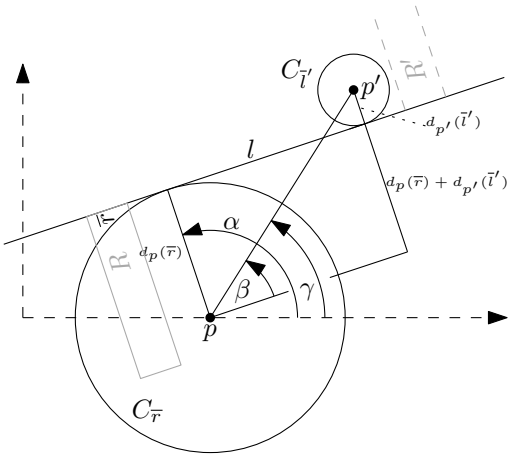


(d) **End** of a possible conflict: p lies to the **left** of $\overline{r}$, p' lies to the **left** of $\overline{l}'$.

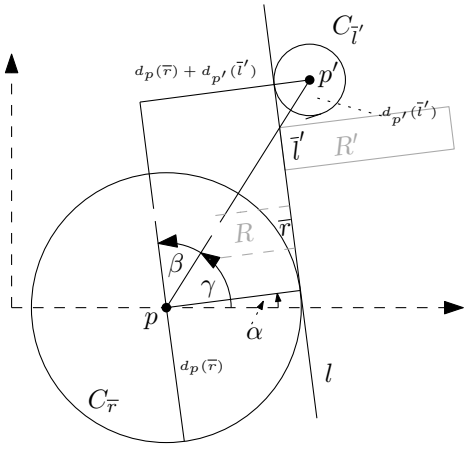
Figure 15.9: Considering outer tangents it shows the conflict events for the right edge of R and the left edge of R'.

the y-axis of the common coordinate system (see Figure 15.11a). More in detail we take the y-axis if the slope γ of $\overline{pp'}$ lies between $\frac{\pi}{2} - \epsilon$ and $\frac{\pi}{2} + \epsilon$ or between $\frac{3\pi}{2} - \epsilon$ and $\frac{3\pi}{2} + \epsilon$ for a pre-defined ϵ with $\frac{\pi}{2} > \epsilon > 0$. Otherwise we project the vertices on the x-axis. Both the projected vertices of R and the projected vertices of R' each form an interval I and I' on the chosen axis. It is easy to see that both intervals intersect if and only if α is a real conflict event, namely if we continue rotating R and R', then either the intersection of both rectangles begins or ends.

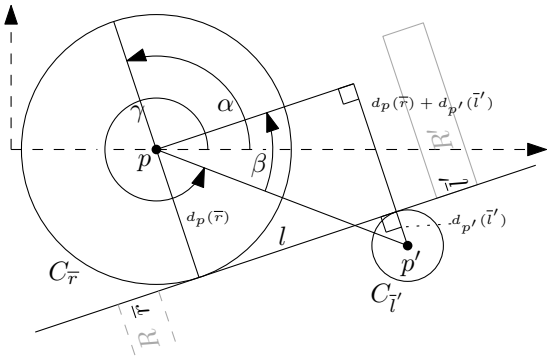
Creating a Sequence of Conflict Events: Now we explain how one can gain a consistent sequence of conflict events for two anchored rectangles R and R' with anchors p and p', that is, a sequence of conflict events describing all conflicts between R and R' consecutively, such that conflict events describing the beginning of a conflict (*start event*) and events describing the end of a conflict (*end event*) occur alternately. In particular we require that such a sequence begins with a start event and ends with an end event of a conflict:



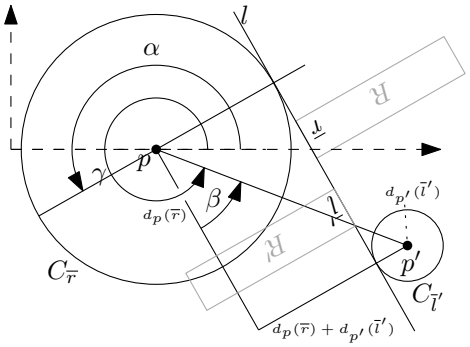
(a) **Begin** of a possible conflict: p lies to the **right** of $\bar{r}$, p' lies to the **left** of $\bar{l}'$.



(b) **End** of a possible conflict: p lies to the **right** of $\bar{r}$, p' lies to the **left** of $\bar{l}'$.

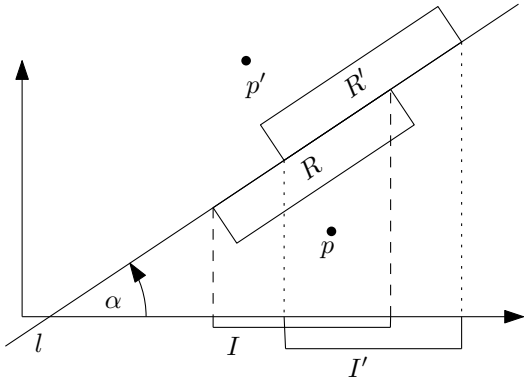


(c) **Begin** of a possible conflict: p lies to the **left** of $\bar{r}$, p' lies to the **right** of $\bar{l}'$.

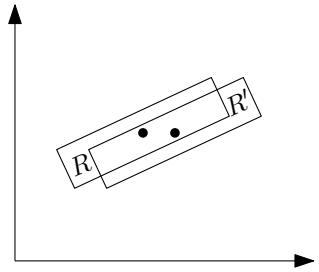


(d) **End** of a possible conflict: p lies to the **left** of $\bar{r}$, p' lies to the **right** of $\bar{l}'$.

Figure 15.10: Considering inner tangents it shows the conflict events for the right edge of R and the left edge of R'.



(a) Projecting vertices on the x-axis.



(b) R and R' are for the whole rotation 2π in conflict.

Figure 15.11: Checking for real conflicts.

Definition 15.1 (Consistent Sequence of Conflict Events)**.** *A sequence $c_1, \ldots, c_k$ of conflict events is called* consistent *if*

1. *c_1 is a start event and c_k is an end event and*
2. *for all $1 < i \leq k$, if c_{i-1} is a start event then c_i is an end event and vice versa and*
3. *there are no two conflict events of the same angle.*

We apply the following approach on R and R': First we determine which main-case we have to consider (see Table 15.1) and then we compute the possible conflict events induced by the edges of R and R'. Afterwards we check for the real conflict events (as described above) and merge those into one non-decreasing sequence sorted by the angle of the conflict events. If a start event c_1 and an end event c_2 have the same angle, we require that c_1 occurs before c_2 in the sequence. Obviously, the resulting sequence contains at most eight events, because for each edge of R we can find at most two conflict events.

In the next step we apply the following rules on the sequence such that we apply each rule until it is not applicable anymore and not till then we apply the next rule:

1. If two start events correspond to the same angle α, then remove one of both start events from the sequence.
2. If a start event and an end event correspond to the same angle α, then remove both from the sequence.
3. If the sequence begins with an end event, then add the start event $\alpha = 0$ to the sequence.
4. If the sequence ends with a start event, then add the end event $\alpha = 2\pi$ to the sequence.

We do not explain explicitly how to implement those rules, but one can do this efficiently in one iteration using a stack. Obviously, the remaining sequence is consistent and still describes all conflicts correctly.

We still have to describe the case that the sequence is empty. In that case we cannot always return that there is no conflict, but we first have to check whether the rectangles are in conflict for the whole time (see Figure 15.11b). To that end we assume a rotation of 0 for both rectangles and check whether they intersect. In the positive case we have found the conflict $[0, 2\pi]$, otherwise we know that they do not have any conflict.

We summarize the results in following theorem:

Theorem 15.1. *For two anchored rectangles R and R' we can compute a consistent sequence $(c_1, \ldots, c_k)$ of conflict events with $k \leq 10$ describing all conflicts between R and R' correctly in $\mathcal{O}(1)$ time.*

In particular the angle intervals $[c_1, c_2], [c_3, c_4], \ldots, [c_{k-1}, c_k]$ are pairwise disjoint and describe all conflicts between R and R'.

Proof. The two rectangles R and R' must satisfy exactly one case as described in Table 15.1. For all cases it is true, that each edge of R induces at most two conflict events (one start event and one end event). As R consists of four edges, we can find at most eight conflict events for R and R'. Since we also add the events 0 and 2π in order to make the sequence consistent when it begins with an end event or ends with a start event, we obtain ten events. □

15.2 Labels in Conflict

Based on the previous section computing the conflicts between all given labels L is quite easy. For each pair $\{l, l'\} \in \binom{L}{2}$ we compute the consistent sequence $S{=}(c_1, \ldots, c_k)$ of conflict events in $\mathcal{O}(1)$ time as described in Theorem 15.1. Then each pair (c_i, c_{i+1}) with $i \bmod 2 = 0$ describes one conflict of l and l', so that

$$C(l, l') = \{[c_1, c_2], [c_3, c_4] \ldots, [c_{k-1}, c_k]\}$$

Lemma 15.1. *Let l and l' be two labels, then the set $C(l, l')$ contains at most five intervals.*

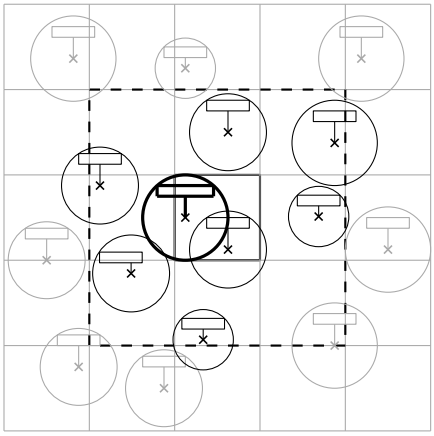
Proof. Follows directly from Theorem 15.1. □

Obviously, we need $\mathcal{O}(n^2)$ time to compute all conflicts between all labels and in fact in the worst case we cannot speed this running time up, because it can happen that all labels are mutually in conflict:

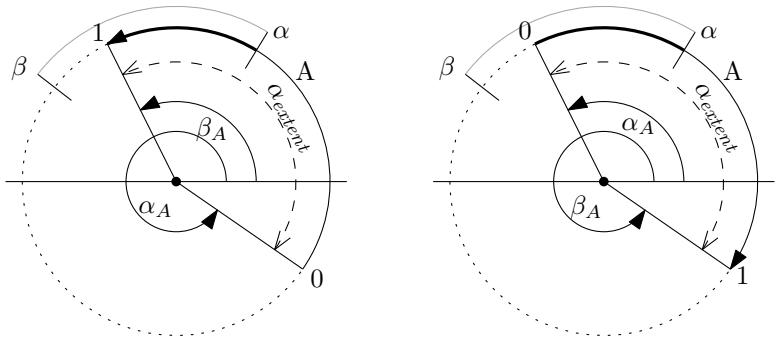
Theorem 15.2. *Given a set L of labels, then we can compute the set $C(L)$ of conflicts between those labels in $\mathcal{O}(n^2)$ time. Further $C(L)$ contains $\mathcal{O}(n^2)$ conflicts.*

Although the worst case time is quadratic, for typical use cases one can improve the running time as the following information box shows.

Implementation Details 15.1. In order to increase the performance for checking for conflicts, there are several well-known solutions as grids and quad-trees applicable that utilize the locality of conflicts and the fact that labels are normally spread over the whole map.
Even though quad-trees often are the better solution than grids, we have decided to implement a grid in order to minimize the time of implementation. The main idea is that the map is underlaid with a homogeneous grid of pre-defined mesh size. Then each label $l \in L$ is related to that cell of the grid that contains the anchor of l. (If the anchor lies on the horizontal border of two cells it is contained in the upper cell and if the anchor lies on the vertical border of two cells it is contained in the left cell.)
Let C_l be the smallest enclosing circle of a label $l \in L$ with center at the anchor of l and let C_{max} denote for all circles C_l with $l \in L$ the largest circle of those. If we then choose the mesh size of the grid at least twice as large as the radius of C_{max}, we know that for a label l we only have to check the labels which are contained in the same cell as l or in one of the neighbor cells:



Further, if we check for two labels l and l' whether they have conflicts, we can first check whether C_l and $C_{l'}$ intersect. If this is not the case, we know that there cannot be any conflicts between both labels. Consequently, we do not need to do a detailed check.



(a) A is counterclockwise oriented.

(b) A is clockwise oriented.

Figure 15.12: Illustration of angles and positions. Both figures show the same arc A and the same interval $[\alpha, \beta]$, but as the orientation of A is different the start and end points of A are also different. The bold sub-arc of A illustrates the positions of A the interval $[\alpha, \beta]$ is mapped on.

15.3 Visible Labels and Visible Conflicts

In this section we explain in more detail how we determine the visibility of labels and conflicts regarding a given viewport $\mathcal{VP}$ and its trajectory T: That means for each label or conflict o we look for a sequence $\Phi_T^{\mathcal{VP}}(o) = ([x_1, y_1], \ldots, [x_k, y_k])$ that describes the visibilities of that entity o regarding $\mathcal{VP}$. For example for a label l the interval $[x_1, y_1]$ means that l is visible within $\mathcal{VP}$ from position x_1 to position y_2.

At first we explain how an angle interval $[\alpha, \beta] \subseteq [0, 2\pi]$ can be translated into an interval $[pos_1, pos_2]$ of local positions for a single arc. Then we use that approach for computing the visibilities of labels and conflicts between labels.

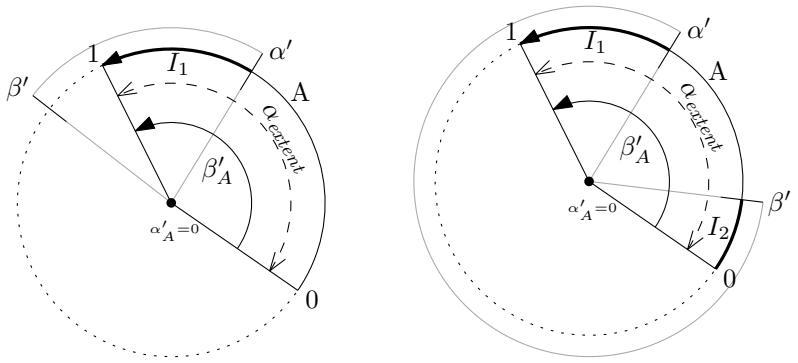
15.3.1 Relation Between Angles and Positions

Assume for this part that we are given an arbitrary arc A and an arbitrary interval $[\alpha, \beta] \subseteq [0, 2\pi]$, then we want to find for $[\alpha, \beta]$ a corresponding interval $[pos_1, pos_2] \subseteq [0, 1]$ of local positions regarding A that spans the same sub-arc of A as $[\alpha, \beta]$ (see Figure 15.12). To that end let α_A be the start angle of A and let β_A be the end angle of A, so that A sweeps the angle α_{extent}. Recall that α_A corresponds to the position 0 of A and β_A corresponds to the position 1 of A.

We distinguish the two cases, whether A is counterclockwise oriented or clockwise oriented, because in the former case going from α_A to β_A along the spanned sub-arc of A the angle increases and it decreases in the latter case.

If A is counterclockwise oriented, we first normalize the angles regarding the start angle α_A of A by subtracting α_A from all angles. In particular we gain the new angles $\alpha' = \alpha - \alpha_A$ and $\beta' = \beta - \alpha_A$ and A then starts at the angle of 0 and ends at an angle of $\beta'_A = \beta_A - \alpha_A$. Further, we assume that the new angles are normalized, that is, they lie in $[0, 2\pi]$.

In the case that $\alpha' \leq \beta'$ (see Figure 15.13a) we can easily derive an interval describing the



(a) $\alpha' \leq \beta'$ (b) $\alpha' > \beta'$

Figure 15.13: Illustration of angles and positions. Both figures show the same counterclockwise oriented arc A, but they differ in the given interval $[\alpha, \beta]$. In particular in the second case the interval $[\alpha, \beta]$ must be split into two intervals.

local positions and the sub-arc of A spanned by $[\alpha', \beta']$:

$$I_1 = \begin{cases} [\max(0, \frac{\alpha'}{\alpha_{extent}}), \min(1, \frac{\beta'}{\alpha_{extent}})] & \alpha' \in [0, \beta'_A] \text{ or } \beta' \in [0, \beta'_A] \\ \emptyset & \text{otherwise} \end{cases}$$

Note that if neither α' nor β' lies within $[0, \beta'_A]$, then $[\alpha, \beta]$ does not span any sub-arc of A. In the case that $\alpha' > \beta'$ (see Figure 15.13b) we check whether at least one angle of α' and β' lies within $[0, \beta'_A]$. If that is the case we return for each angle that lies within $[0, \beta'_A]$ one interval:

$$I_1 = \begin{cases} [\frac{\alpha'}{\alpha_{extent}}, 1] & 0 \leq \alpha' \leq \beta'_A \\ \emptyset & \text{otherwise} \end{cases} \qquad I_2 = \begin{cases} [0, \frac{\beta'}{\alpha_{extent}}] & 0 \leq \beta' \leq \beta'_A \\ \emptyset & \text{otherwise} \end{cases}$$

In the case that neither α' nor β' lie within $[0, \beta'_A]$, we return the interval $I_1 = [0, 1]$.

If A is clockwise oriented, we first assume A to be counterclockwise oriented, that is, the start angle becomes the end angle and vise versa (see Figure 15.12). We then apply the approach as described above in order to gain the Intervals $I_1 = [pos_1, pos_2]$, $I_2 = [pos_3, pos_4]$. Without loss of generality we assume that they are defined. We then redefine both intervals in order to comprise that A is in fact clockwise oriented: I_1:=$[1 - pos_2, 1 - pos_1]$ and I_2:=$[1 - pos_4, 1 - pos_3]$.

We denote the procedure that converts angles $[\alpha, \beta] \subseteq [0, 2\pi]$ into local positions for an arc A by `convertAnglesToPositions` and assume that it returns a sequence of only non-empty position intervals as described above. If the given angle interval does not span any part of the given arc, then it returns an empty sequence.

Up to now those intervals contain local positions for a certain arc, but normally we consider a trajectory T that consists of several arcs, so that we often need global positions. We therefore extend the function lg_T of Chapter 14 canonically to sequences of local position intervals:

$$\text{lg}_T(A, [pos, pos']) := [\text{lg}_T(A, pos), \text{lg}_T(A, pos')]$$
$$\text{lg}_T(A, ([pos_1, pos'_1], \ldots, [pos_k, pos'_k])) := (\text{lg}_T(A, [pos_1, pos'_1]), \ldots, \text{lg}_T(A, [pos_k, pos'_k]))$$

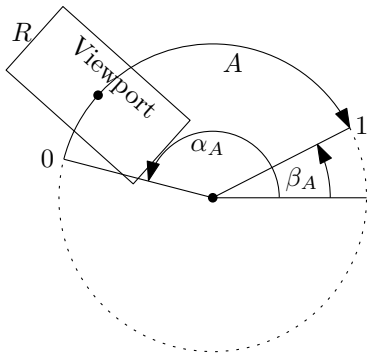


Figure 15.14: Restricted to one arc the viewport can be seen as an anchored rectangle.

15.3.2 Visibility of Labels:

Now we explain how we can gain the visibilities for a label l regarding a viewport $\mathcal{VP}$ and its trajectory T with arcs $A_1, \ldots, A_m$. We again first consider a single arc A_i and describe how we can gain the visibilities of l regarding that arc and $\mathcal{VP}$. To that end let R be the anchored rectangle $R(\mathcal{VP}, A_i)$ as defined in Chapter 14. Consequently, if we rotate R around its anchor there is an angle interval $[\alpha_{A_i}, \beta_{A_i}]$ which corresponds to A_i, namely α_{A_i} is the start angle of A_i and β_{A_i} is the end angle of A_i (see Figure 15.14).

First we compute the sequence of conflicts $[\alpha_1, \beta_1], \ldots, [\alpha_k, \beta_k]$ between l and R by means of the approach described in Section 15.1. Then we translate each interval into the local positions of A_i using the procedure `convertAnglesToPositions` and merge the results into one sequence $\Phi_{A_i}^{\mathcal{VP}}(l)$ after converting the local into global positions regarding T and sorting the intervals in non-decreasing order regarding the first coordinate of the intervals:

$$\Phi_{A_i}^{\mathcal{VP}}(l) = \text{sort}(\uparrow, \bigcup_{j=1}^{k} \text{lg}_T(A_i, \texttt{convertAnglesToPositions}([\alpha_j, \beta_j], A_i)))$$

As the angles $[\alpha_1, \beta_1], \ldots, [\alpha_k, \beta_k]$ are disjoint (Theorem 15.1), the intervals of $\Phi_{A_i}^{\mathcal{VP}}(l)$ must also be disjoint because we only convert the intervals but we do not change their structure.

We apply that procedure on all arcs $A_1, \ldots, A_m$ in order to gain $\Phi_{A_1}^{\mathcal{VP}}(l), \ldots, \Phi_{A_m}^{\mathcal{VP}}(l)$ and then we assemble them to one sequence $\Phi_T^{\mathcal{VP}}(l)$ maintaining their order:

$$\Phi_T^{\mathcal{VP}}(l) = \Phi_{A_1}^{\mathcal{VP}}(l) + \ldots + \Phi_{A_m}^{\mathcal{VP}}(l)$$

Finally, we iterate through $\Phi_T^{\mathcal{VP}}(l)$ once in order to merge overlapping intervals: For two consecutive intervals I=$[pos_1, pos_2]$ and I'=$[pos'_1, pos'_2]$ we check whether they overlap. If that is the case, we replace both by the interval $[pos_1, pos'_2]$. Note that I and I' refer to positions on different arcs A and A', so that they can only overlap in one point, namely in the transition of A and A'. Thus, we obtain $pos_1 < pos_2 \leq pos'_1 < pos'_2$. On that account $[pos_1, pos'_2]$ describes both visibilities I and I' completely and we do not loose visibility when replacing I and I' with the new interval. Consequently, $\Phi_T^{\mathcal{VP}}(l)$ is the sequence of visibilities of l as described in Definition 14.1.

For the running time we consider the following lemma:

Lemma 15.2. *Let l be an arbitrary label and let A be an arbitrary arc of a given trajectory T with viewport $\mathcal{VP}$, such that the rectangles l and $\mathcal{VP}$ have the conflicts $[\alpha_1, \beta_1], \ldots, [\alpha_k, \beta_k]$*

regarding A (in angles), then the sequence

$$\Phi_A^{\mathcal{VP}}(l) = \text{sort}(\uparrow, \bigcup_{j=1}^{k} \text{lg}_T(A, \texttt{convertAnglesToPositions}([\alpha_j, \beta_j], A)))$$

can contain at most five intervals.

Proof. From Theorem 15.1 we know that there are at most ten conflict events between l and $\mathcal{VP}$. Since two of each form one conflict there can be at most five conflicts. □

Corollary 15.1. *The sequence $\Phi_T^{\mathcal{VP}}(l)$ contains at most $4 \cdot m$ intervals.*

Due to Theorem 15.1 we need only $\mathcal{O}(1)$ time to compute the conflicts between R and l and according to the previous lemma we can sort $\Phi_{A_i}^{\mathcal{VP}}(l)$ for an arc A_i in $\mathcal{O}(1)$ time. Consequently, we need $\mathcal{O}(m)$ time for constructing $\Phi_T^{\mathcal{VP}}(l)$.

When we apply that procedure on all labels we gain the following theorem:

Theorem 15.3. *Given a set $L = \{l_1, \ldots, l_n\}$ of labels and a viewport $\mathcal{VP}$ with trajectory $T = (A_1, \ldots, A_m)$, then we can compute the sequences $\Phi_T^{\mathcal{VP}}(l_1), \ldots, \Phi_T^{\mathcal{VP}}(l_n)$ of visibilities of the labels in L in $\mathcal{O}(m \cdot n)$ time, such that they satisfy Definition 14.1.*

15.3.3 Visibility of Conflicts:

In this part we assume that we are given a set L of labels and a viewport $\mathcal{VP}$ with trajectory $T = (A_1, \ldots, A_m)$. We want to compute the visibilities of all conflicts $C(L)$ between the given labels. To that end we apply a very similar approach as used for the labels in order to compute the visibilities, but first we show the next lemma.

Lemma 15.3. *Given a viewport $\mathcal{VP}$ with its trajectory $T = (A_1, \ldots, A_m)$ and a conflict $c = (l, l')$ between two labels l and l', then the following statement is true:*

The conflict $c = (l, l')$ is visible in $\mathcal{VP}$ at a position pos of T if and only if both labels l and l' are visible in $\mathcal{VP}$ at pos.

Proof. Recall for the proof that a conflict of two labels l and l' at a position *pos* of T is defined as a non-empty intersection between both labels. "⇒": Obviously, when the conflict is visible then both labels must be visible. "⇐": As l and l' are axis-aligned rectangles, this direction holds also. □

Analogously to labels, we again consider only one conflict $c=\{[\alpha, \beta], l, l'\} \in C(L)$ and describe how we gain the visibilities for that conflict. To that end we proceed for each arc A_i (with $1 \leq i \leq m$) as follows:

First we translate $[\alpha, \beta]$ into a sequence of local position intervals by means of the procedure `convertAnglesToPositions` and then we convert those local positions regarding A_i into global positions regarding T and sort those intervals in non-decreasing order regarding the first coordinate of the intervals:

$$S_{A_i}^{[\alpha,\beta]} = \text{sort}(\uparrow, \text{lg}_T(A_i, \texttt{convertAnglesToPositions}([\alpha, \beta], A_i)))$$

After we have done this for all arcs, we concatenate the resulting sequences $S_{A_1}^{[\alpha,\beta]}, \ldots, S_{A_m}^{[\alpha,\beta]}$ into one long sequence $S_{[\alpha,\beta]}$ maintaining the order:

$$S_{[\alpha,\beta]} = S_{A_1}^{[\alpha,\beta]} + \ldots + S_{A_m}^{[\alpha,\beta]}$$

Analogously to computing visibilities, we merge all overlapping intervals in $S_{[\alpha,\beta]}$, that is, if two intervals $I=[pos_1, pos_2]$ and $I'=[pos'_1, pos'_2]$ of $S_{[\alpha,\beta]}$ overlap, then we replace both by the single interval $[pos_1, pos'_2]$. We can again argue that after applying this step the sequence $S_{[\alpha,\beta]}$ contains only non-overlapping intervals sorted in increasing order (for a detailed description see the analogous part for computing the visibilities of labels).

Lemma 15.4. *Given two labels l and l' with conflict c and a viewport $\mathcal{VP}$ with trajectory $T = (A_1, \ldots, A_m)$, then the sequence $S_{[\alpha,\beta]}$ as described above contains at most $\mathcal{O}(m)$ intervals.*

Proof. We first show for each arc A_i that the set $S_{A_i}^{[\alpha,\beta]}$ can contribute at most two intervals, because then the statement follows directly: Have a detailed look at the description of the procedure `convertAnglesToPositions`. By definition of the procedure the passed interval $[\alpha, \beta]$ is split into at most two intervals. Thus, according to the definition of $S_{A_i}^{[\alpha,\beta]}$ it also contains at most two intervals. □

So far we have not made use of the visibility $\Phi_T^{\mathcal{VP}}(l)$ and $\Phi_T^{\mathcal{VP}}(l')$ of l and l' which we have introduced in the previous part. Consequently, the intervals in $S_{[\alpha,\beta]}$ can be oversized. We therefore compute the intersection of $S_{[\alpha,\beta]}$, $\Phi_T^{\mathcal{VP}}(l)$ and $\Phi_T^{\mathcal{VP}}(l')$:

$$\Phi_T^{\mathcal{VP}}(c) = S_{[\alpha,\beta]} \cap \Phi_T^{\mathcal{VP}}(l) \cap \Phi_T^{\mathcal{VP}}(l')$$

where the intersection $S_1 \cap S_2$ of two sequences of intervals is defined as:

$$S_1 \cap S_2 = \{[p_1, q_1] \cap [p_2, q_2] \mid [p_1, q_1] \in S_1 \text{ and } [p_2, q_2] \in S_2\}$$

Summarizing we can say that the sequence $\Phi_T^{\mathcal{VP}}(c)$ describes the visibility of the given conflict $c = \{[\alpha, \beta], l, l'\}$ regarding $\mathcal{VP}$ and T correctly, because

1. $S_{[\alpha,\beta]}$ is the sequence of intervals describing the angular conflict interval $[\alpha, \beta]$ by means of positions.
2. $\Phi_T^{\mathcal{VP}}(l)$ and $\Phi_T^{\mathcal{VP}}(l')$ are the sequences of intervals describing the visibilities of l and l' by means of positions.
3. $\Phi_T^{\mathcal{VP}}(c)$ is defined as the intersection $S_{[\alpha,\beta]} \cap \Phi_T^{\mathcal{VP}}(l) \cap \Phi_T^{\mathcal{VP}}(l')$, that is, we only consider positions that are in common with all three sequences.
4. Lemma 15.3 holds.

It remains to show how we can efficiently compute the intersection $S = S_1 \cap S_2$ of two sequences $S_1 = ([p_1, q_1], \ldots, [p_u, q_u])$ and $S_2 = ([s_1, t_1], \ldots, [s_v, t_v])$ that contain disjoint intervals sorted in increasing order.

For that purpose we apply one sweep over both sequences simultaneously: Let i be the index of the currently considered interval in S_1 and let j be the index of the currently considered interval in S_2. We start with $i = 1$ and $j = 1$ and check whether $[p_i, q_i] \cap [s_j, t_j] \neq \emptyset$. If that is the case we add $[p_i, q_i] \cap [s_j, t_j]$ to the resulting sequence S. Afterwards we increase i by one if $q_i \leq t_j$ and otherwise we increase j by one. We repeat the whole step until $i > u$ or $j > v$. Figure 15.15 illustrates the procedure.

Consequently, we need $\mathcal{O}(|u| + |v|)$ time to compute $S = S_1 \cap S_2$. Based on that the next lemma makes a statement about the running time for computing $\Phi_T^{\mathcal{VP}}(c)$.

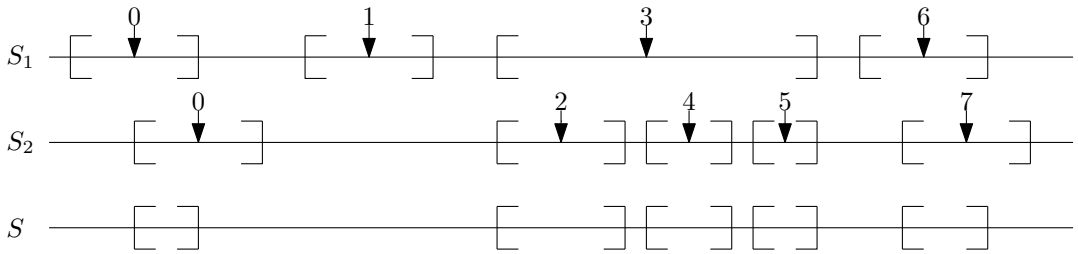


Figure 15.15: Illustrates the intersection of two sequences S_1 and S_2 containing intervals. S is the resulting interval and the numbers above the intervals indicate after which iteration step the index variables i and j are set to that particular interval.

Lemma 15.5. *Given two labels l and l' having the conflict $c = \{[\alpha, \beta], l, l'\}$, a viewport $\mathcal{VP}$ with trajectory $T = (A_1, \ldots, A_m)$ and the visibilities $\Phi_T^{\mathcal{VP}}(l)$ and $\Phi_T^{\mathcal{VP}}(l')$ of both labels, then we can compute the visibilities $\Phi_T^{\mathcal{VP}}(c)$ of c in $\mathcal{O}(m)$ time.*

Proof. From Corollary 15.1 we know that $\Phi_T^{\mathcal{VP}}(l)$ and $\Phi_T^{\mathcal{VP}}(l')$ contain at most $\mathcal{O}(m)$ visibilities and from Lemma 15.3 follows that $S_{[\alpha,\beta]}$ has at most $\mathcal{O}(m)$ elements. Thus, we need $\mathcal{O}(m)$ time to compute the intersection $\Phi_T^{\mathcal{VP}}(c) = S_{[\alpha,\beta]} \cap \Phi_T^{\mathcal{VP}}(l) \cap \Phi_T^{\mathcal{VP}}(l')$. □

So far we have considered only one conflict $c = \{[\alpha, \beta], l, l'\}$ of $C(L)$, but we need to apply the whole procedure on all conflicts contained in $C(L)$ which leads to the following theorem:

Theorem 15.4. *Given a set $L = \{l_1, \ldots, l_n\}$ of labels with conflicts $C(L) = \{c_1, \ldots, c_k\}$ and a viewport $\mathcal{VP}$ with its trajectory $T = (A_1, \ldots, A_m)$, then we can compute the sequences $\Phi_T^{\mathcal{VP}}(c_1)$, …, $\Phi_T^{\mathcal{VP}}(c_k)$ of visibilities of the conflicts in $\mathcal{O}(k \cdot m)$ time, such that they satisfy Definition 14.2.*

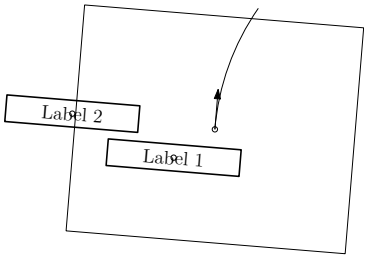
15.4 Area Based Visibility for Labels

So far we have defined the visibility of labels regarding the duration for which they stay within the given viewport. Since we want to maximize the overall visibility over all labels, it can occur that only partly shown labels are preferred over labels that are completely shown, when the former one stays longer within the viewport than the latter one. Figure 15.16 shows such an example and its drawback: Often it is better to set the label with larger intersection area inactive than the other. In the worst case this leads to the constellation that the user cannot read any of both labels, because one label is set inactive and the other label is only fractionally contained in the viewport. On that account we present in this section how the visibility of labels could be redefined using the intersection area between labels and viewport. To that end we first explain how the intersection area of two anchored rectangles can be computed and then in the second part we explain how this intersection area can be used for redefining the visibility of labels.

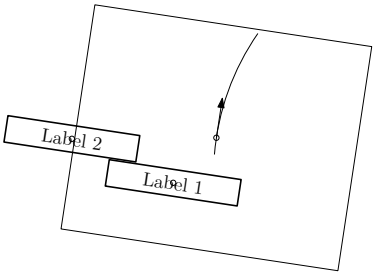
Implementation Details 15.2. So far this part has not been implemented, yet.

15.4.1 Anchored Rectangles and Their Conflict Area

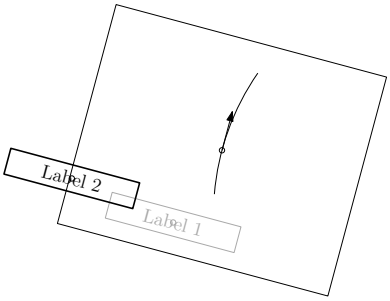
Assume that we are given two anchored rectangles $R = (\bar{l}, \bar{r}, \bar{b}, \bar{t})$ and $R' = (\bar{l}', \bar{r}', \bar{b}', \bar{t}')$ with anchors p and p' and the angle interval $[\alpha, \beta]$, then we explain now how the intersection area



(a) 1. Position: Both labels are active.



(b) 2. Position: A conflict between labels arises.



(c) 3. Position: The label with less intersection area with the viewport than the other has won the conflict.

Figure 15.16: Restricted visibility of labels. A partly visible label may cover another label that is fully visible within the viewport (black label=active label, gray label=inactive label).

of R and R' can be computed for the rotation that takes place in $[\alpha, \beta]$. More formally we explain how to gain

$$A(R, R', \alpha, \beta) = \int_{\alpha}^{\beta} A(R, R', \rho) d\rho$$

where $A(R, R', \rho)$ denotes the intersection area between R and R' for the common rotation angle ρ. To that end we consider the rotation of R and R' within the angle interval $[\alpha, \beta]$ and split that rotation into five different phases. Each phase is characterized by the number of edges of R involved in the intersection of R and R'. Figure 15.17 shows all possible types of a phase:

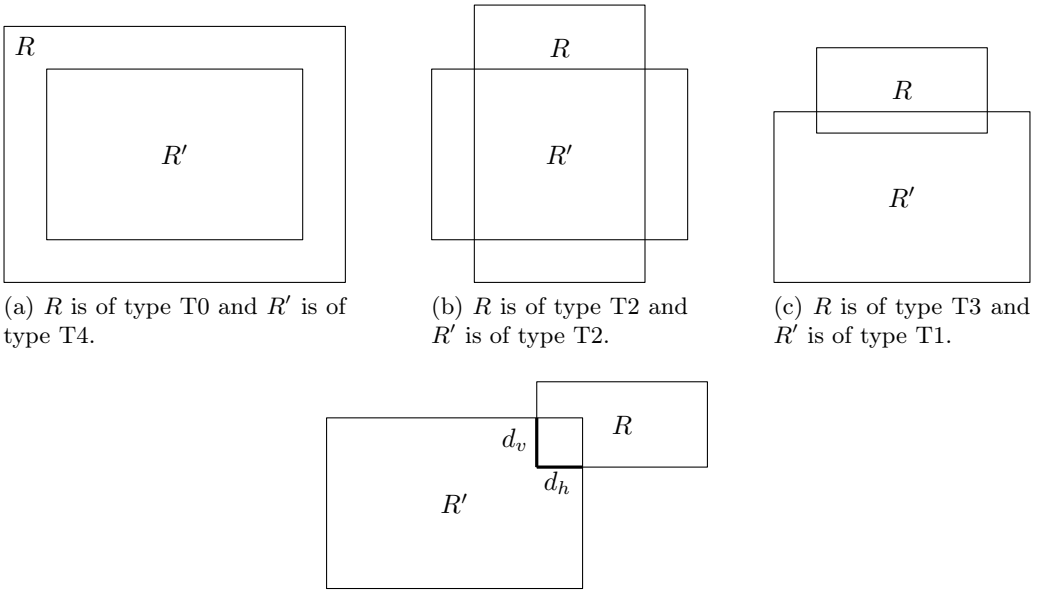
T0: The edges of R do not have any points in common with R'.

T1: One edge of R is involved in the intersection of R and R'.

T2: Two edges of R are involved in the intersection of R and R', then there are six sub-cases: The edges $\{\overline{r}, \overline{b}\}$, $\{\overline{r}, \overline{t}\}$, $\{\overline{l}, \overline{b}\}$, $\{\overline{l}, \overline{t}\}$, $\{\overline{t}, \overline{b}\}$ or $\{\overline{l}, \overline{r}\}$ are involved.

T3: Three edges of R are involved in the intersection of R and R', then there are four sub-cases: The edges $\{\overline{r}, \overline{b}, \overline{l}\}$, $\{\overline{r}, \overline{t}, \overline{l}\}$, $\{\overline{t}, \overline{r}, \overline{b}\}$ or $\{\overline{t}, \overline{l}, \overline{b}\}$ are involved.

T4: All four edges of R are involved in the intersection of R and R'.



(a) R is of type T0 and R' is of type T4.

(b) R is of type T2 and R' is of type T2.

(c) R is of type T3 and R' is of type T1.

(d) R is of type T2 and R' is of type T2.

Figure 15.17: Intersection types. The figure shows all possible intersection types that can occur for two intersecting rectangles. The difference between 15.17b and 15.17d is that in the former one edges of the same orientation are involved, while in the latter one edges of different orientation are involved.

The individual phases for the intersection of two anchored rectangles can be obtained by adapting the approach of Section 15.1 for computing conflict events between anchored rectangles. As this is mostly a technical detail, we only sketch the adaption shortly: So far we only considered opposite side edges of anchored rectangles when we computed the individual conflict events. We therefore gain events for the beginning and the end of the intersection rotating both rectangles simultaneously. If we also consider two edges representing the same sides (e.g. the top edges of R and R'), we then obtain events describing the different phases we are looking for.

Ordered by their complexity we now describe how to handle the different types of a phase. To that purpose we assume that the considered phase spans the angle interval $[\alpha, \beta]$ and that R has the width w and the height h.

Type T0: For this type we only have to check, whether R contains R'. If that is the case the intersection area is obviously

$$A_0(R, R', \alpha, \beta) = \int_{\alpha}^{\beta} w \cdot h \; d\rho = [w \cdot h \cdot \rho]_{\rho=\alpha}^{\rho=\beta}$$

Otherwise, we set $A_0(R, R', \alpha, \beta) = 0$.

Type T4: For this type we know that R is completely contained in R', so that the intersection area is:

$$A_4(R, R', \alpha, \beta) = \int_{\alpha}^{\beta} w \cdot h \; d\rho = [w \cdot h \cdot \rho]_{\rho=\alpha}^{\rho=\beta}$$

Type T1: If only one edge is involved in the intersection, the edge must intersect two edges of R'. Consequently, R' has two edges that are involved in the intersection. We therefore can switch the roles between R and R' and can treat it as type T3:

$$A_1(R, R', \alpha, \beta) = A_3(R', R, \alpha, \beta)$$

Type T3: If three edges of R enclose the intersection area, one of the three edges must be completely contained in R'. We denote that edge by e and distinguish whether it is a horizontal or vertical edge.

The edge e is a horizontal edge: At first we discuss the case that e is the top edge $\bar{t}$ of R and later on, by means of symmetries we adapt the results to the case that e is the bottom edge $\bar{b}$ of R. So assume that e is the top edge $\bar{t}$ of R, then the intersection is enclosed by the edges $\bar{t}$, $\bar{r}$, $\bar{l}$ of R and the edge $\bar{b}'$ of R' (see Figure 15.18a, black labels). (Note that for the following the orientation of both rectangles is indicated by the orientation of the label R and R' in the corresponding figures.)

In order to compute the gray shaded intersection of R and R' we need the length of the segment d_1, because the intersection is determined by $d_1 \cdot w$. We cannot directly compute d_1, but we can assemble it by different distances and angles that we know:

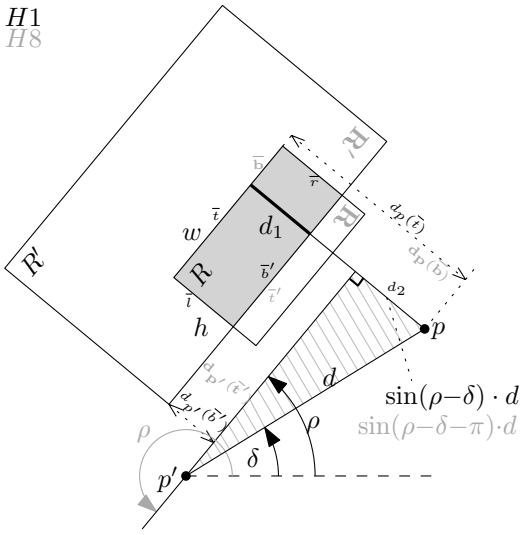
1. The angle δ describes the fixed slope of the line through p and p'.
2. The angle ρ coincides with the rotation of R and R' and is therefore the angle we use in the integral.
3. The distance $d_p(\bar{t})$ is pre-defined by the relation between p and R.[2]
4. The distance $d_{p'}(\bar{b}')$ is pre-defined by the relation between p' and R'.
5. The distance d between p and p'.

For the analysis we have to distinguish the following four sub-cases describing the position of p and p' relative to $\bar{t}$ and $\bar{b}'$, respectively:

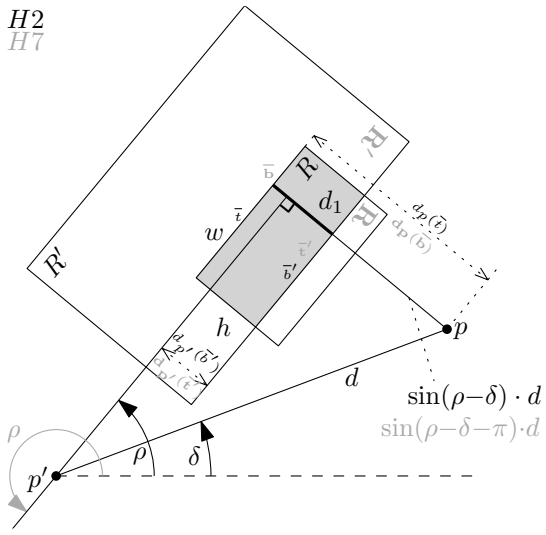
$H1$	p lies to the **right** of $\bar{t}$	p' lies to the **right** of $\bar{b}'$	Figure 15.18a, black
$H2$	p lies to the **right** of $\bar{t}$	p' lies to the **left** of $\bar{b}'$	Figure 15.18b, black
$H3$	p lies to the **left** of $\bar{t}$	p' lies to the **right** of $\bar{b}'$	Figure 15.18c, black
$H4$	p lies to the **left** of $\bar{t}$	p' lies to the **left** of $\bar{b}'$	Figure 15.18d, black

In all cases we want to express d_1 by the known terms, because then we can easily compute the area by $d_1 \cdot w$. We only discuss the first sub-case in more detail and present for the others cases corresponding drawings, so that the reader can verify the results.

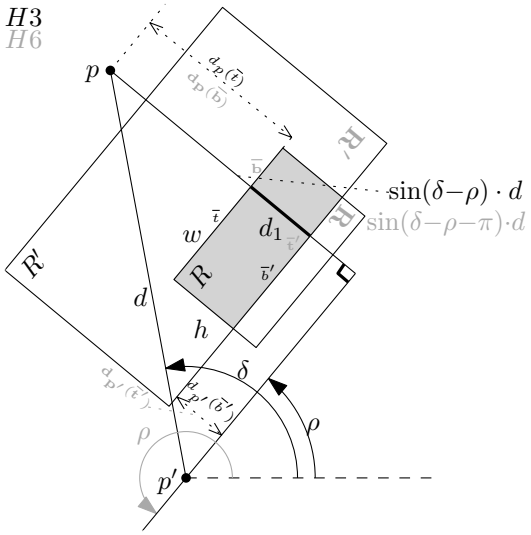
[2]Recall that $d_p(\bar{l})$ denotes the distance between p and the line that extends the line segment $\bar{l}$.



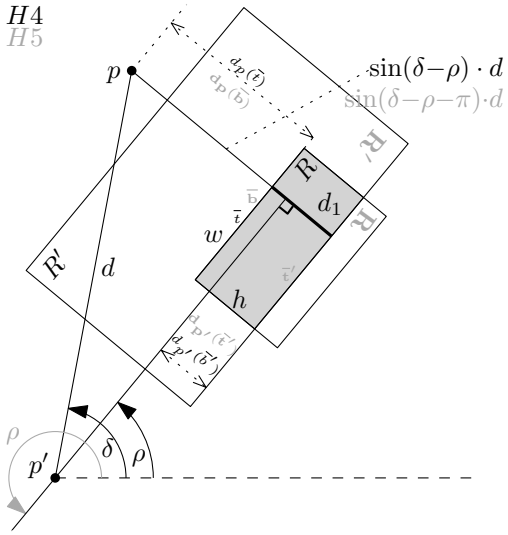
(a) Black: p lies to the **right** of $\overline{t}$ and p' lies to the **right** of $\overline{b}'$. Gray: p lies to the **left** of $\overline{b}$ and p' lies to the **left** of $\overline{t}'$.



(b) Black: p lies to the **right** of $\overline{t}$ and p' lies to the **left** of $\overline{b}'$. Gray: p lies to the **left** of $\overline{b}$ and p' lies to the **right** of $\overline{t}'$.



(c) Black: p lies to the **left** of $\overline{t}$ and p' lies to the **right** of $\overline{b}'$. Gray: p lies to the **right** of $\overline{b}$ and p' lies to the **left** of $\overline{t}'$.



(d) Black: p lies to the **left** of $\overline{t}$ and p' lies to the **left** of $\overline{b}'$. Gray: p lies to the **right** of $\overline{b}$ and p' lies to the **right** of $\overline{t}'$.

Figure 15.18: The rotation phase T2. Shows the case that the edges $\overline{t}$, $\overline{r}$, $\overline{l}$ of R and the edge $\overline{b}'$ of R' enclose the intersection area (black labels) and the case that the edges $\overline{b}$, $\overline{r}$, $\overline{l}$ of R and the edge $\overline{t}'$ of R' enclose the intersection area (gray labels).

So assume that p lies to the right of $\overline{t}$ and p' lies to the right of $\overline{b}'$ (see Figure 15.18a). We first consider the hatched rectangular triangle: We can describe the distance d_2 by means of the angles ρ and δ:

$$\sin(\rho - \delta) = \frac{d_2}{d} \quad \Rightarrow \quad d_2 = \sin(\rho - \delta) \cdot d$$

Then it is easy to see that d_1 is obtained from the distances d_2, $d_p(\overline{t})$ and $d_{p'}(\overline{b}')$:

$$d_1 = d_p(\overline{t}) - d_2 - d_{p'}(\overline{b}') = \underbrace{d_p(\overline{t}) - d_{p'}(\overline{b}')}_{h_1} - \sin(\rho - \delta) \cdot d$$

The following table summarizes the analogous results for all four cases $H1$-$H4$:

$H1$	$d_1 = h_1 - \sin(\rho - \delta) \cdot d$	with $h_1 = d_p(\overline{t}) - d_{p'}(\overline{b}')$
$H2$	$d_1 = h_2 - \sin(\rho - \delta) \cdot d$	with $h_2 = d_p(\overline{t}) + d_{p'}(\overline{b}')$
$H3$	$d_1 = h_3 - \sin(\rho - \delta) \cdot d$	with $h_3 = -d_p(\overline{t}) - d_{p'}(\overline{b}')$
$H4$	$d_1 = h_4 - \sin(\rho - \delta) \cdot d$	with $h_4 = -d_p(\overline{t}) + d_{p'}(\overline{b}')$
Note that $\sin(\delta - \rho) = -\sin(\rho - \delta)$		

Now assume that the considered horizontal edge e is the bottom edge of R (see gray labels in Figure 15.18): This time the intersection is enclosed by the edges $\overline{b}$, $\overline{r}$, $\overline{l}$ of R and the edge $\overline{t}'$ of R'. We again have to distinguish the following cases:

$H5$	p lies to the **right** of $\overline{b}$	p' lies to the **right** of $\overline{t}'$	Figure 15.18d, gray
$H6$	p lies to the **right** of $\overline{b}$	p' lies to the **left** of $\overline{t}'$	Figure 15.18c, gray
$H7$	p lies to the **left** of $\overline{b}$	p' lies to the **right** of $\overline{t}'$	Figure 15.18b, gray
$H8$	p lies to the **left** of $\overline{b}$	p' lies to the **left** of $\overline{t}'$	Figure 15.18a, gray

Obviously, due to symmetry we can easily reuse the previous results: only the names of the segments and the orientation of the anchors have changed but not the structure in general. We therefore gain the following table:

$H5$	$d_1 = h_5 + \sin(\rho - \delta) \cdot d$	with $h_5 = -d_p(\overline{b}) + d_{p'}(\overline{t}')$
$H6$	$d_1 = h_6 + \sin(\rho - \delta) \cdot d$	with $h_6 = -d_p(\overline{b}) - d_{p'}(\overline{t}')$
$H7$	$d_1 = h_7 + \sin(\rho - \delta) \cdot d$	with $h_7 = d_p(\overline{b}) + d_{p'}(\overline{t}')$
$H8$	$d_1 = h_8 + \sin(\rho - \delta) \cdot d$	with $h_8 = d_p(\overline{b}) - d_{p'}(\overline{t}')$
Note that $\sin(\rho - \delta - \pi) = -\sin(\rho - \delta)$		

As $h_1, \ldots, h_8$ are constants, we can treat them equally and can therefore define the parameterized area for h_i (with $1 \leq i \leq 8$):

$$A_{h_i,\delta}(R, R', \alpha, \beta) = \int_{\alpha}^{\beta} w \cdot (h_i + \mathrm{sgn}(i) \cdot \sin(\rho - \delta))\ d\rho = [w \cdot h_i \cdot \rho]_{\rho=\alpha}^{\rho=\beta} - \mathrm{sgn}(i)[\cos(\rho - \delta)]_{\rho=\alpha}^{\rho=\beta}$$

with

$$\mathrm{sgn}(i) = \begin{cases} -1 & 1 \leq i \leq 4 \\ 1 & 5 \leq i \leq 8. \end{cases}$$

The edge e is a vertical edge: Since vertical edges can be handled analogously, we only present the results and the corresponding drawings, so that the reader can verify the results. We again distinguish the two cases that the intersection area is enclosed by

$$\overline{r}, \overline{b}, \overline{t} \text{ of } R \text{ and } \overline{l}' \text{ of } R' \textbf{ or } \overline{l}, \overline{b}, \overline{t} \text{ of } R \text{ and } \overline{r}' \text{ of } R'$$

Further, we distinguish the cases of the position of the anchors, which are also illustrated in Figure 15.19:

$V1$	p lies to the **right** of $\overline{r}$	p' lies to the **right** of $\overline{l}'$	Figure 15.19a
$V2$	p lies to the **right** of $\overline{r}$	p' lies to the **left** of $\overline{l}'$	Figure 15.19b
$V3$	p lies to the **left** of $\overline{r}$	p' lies to the **right** of $\overline{l}'$	Figure 15.19c
$V4$	p lies to the **left** of $\overline{r}$	p' lies to the **left** of $\overline{l}'$	Figure 15.19d
$V5$	p lies to the **right** of $\overline{l}$	p' lies to the **right** of $\overline{r}'$	Figure 15.19d
$V6$	p lies to the **right** of $\overline{l}$	p' lies to the **left** of $\overline{r}'$	Figure 15.19c
$V7$	p lies to the **left** of $\overline{l}$	p' lies to the **right** of $\overline{r}'$	Figure 15.19b
$V8$	p lies to the **left** of $\overline{l}$	p' lies to the **left** of $\overline{r}'$	Figure 15.19a

Then we gain the following results:

$V1$	$d_1 = v_1 - \sin(\rho - \delta - \frac{\pi}{2}) \cdot d$	with $v_1 = d_p(\overline{r}) - d_{p'}(\overline{l}')$
$V2$	$d_1 = v_2 - \sin(\rho - \delta - \frac{\pi}{2}) \cdot d$	with $v_2 = d_p(\overline{r}) + d_{p'}(\overline{l}')$
$V3$	$d_1 = v_3 - \sin(\rho - \delta - \frac{\pi}{2}) \cdot d$	with $v_3 = -d_p(\overline{r}) - d_{p'}(\overline{l}')$
$V4$	$d_1 = v_4 - \sin(\rho - \delta - \frac{\pi}{2}) \cdot d$	with $v_4 = -d_p(\overline{r}) + d_{p'}(\overline{l}')$
$V5$	$d_1 = v_5 + \sin(\rho - \delta - \frac{\pi}{2}) \cdot d$	with $v_5 = -d_p(\overline{r}) + d_{p'}(\overline{l}')$
$V6$	$d_1 = v_6 + \sin(\rho - \delta - \frac{\pi}{2}) \cdot d$	with $v_6 = -d_p(\overline{r}) - d_{p'}(\overline{l}')$
$V7$	$d_1 = v_7 + \sin(\rho - \delta - \frac{\pi}{2}) \cdot d$	with $v_7 = +d_p(\overline{r}) + d_{p'}(\overline{l}')$
$V8$	$d_1 = v_8 + \sin(\rho - \delta - \frac{\pi}{2}) \cdot d$	with $v_8 = +d_p(\overline{r}) - d_{p'}(\overline{l}')$

The main difference to horizontal edges is that there is an offset of $\frac{\pi}{2}$ for the rotation angle. Then we derive the following parameterized integral for v_i (with $1 \leq i \leq 4$):

$$\begin{aligned} A_{v_i,\delta}(R, R', \alpha, \beta) &= \int_{\alpha}^{\beta} h \cdot (v_i + \text{sgn}(i) \sin(\rho - \delta - \frac{\pi}{2}))\ d\rho \\ &= [h \cdot v_i \cdot \rho]_{\rho=\alpha}^{\rho=\beta} - \text{sgn}(i)[\cos(\rho - \delta)]_{\rho=\alpha}^{\rho=\beta} \end{aligned}$$

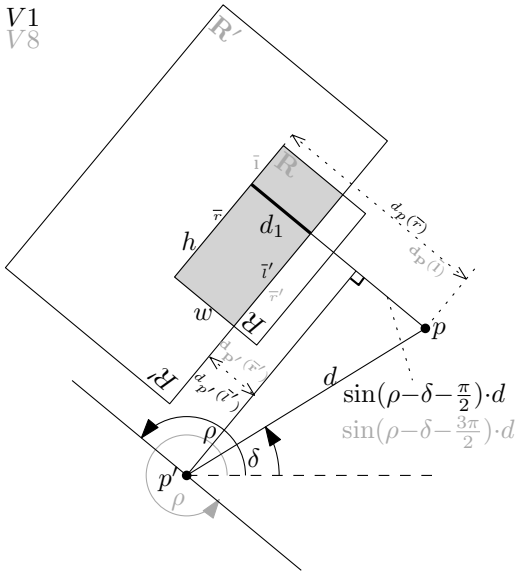
where sgn is identically defined as for the case of horizontal edges:

$$\text{sgn}(i) = \begin{cases} -1 & 1 \leq i \leq 4 \\ 1 & 5 \leq i \leq 8 \end{cases}$$

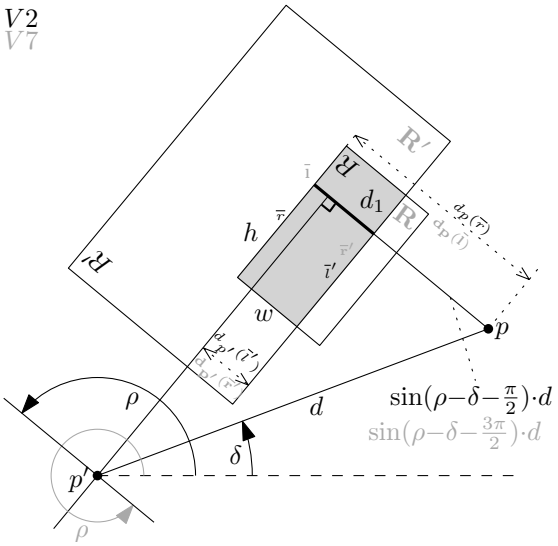
Finally, we define

$$A_3(R, R', \alpha, \beta) := \begin{cases} A_{h_i,\delta}(R, R', \alpha, \beta) & \text{if the edge } e \text{ is a horizontal edge.} \\ A_{v_j,\delta}(R, R', \alpha, \beta) & \text{if the edge } e \text{ is a vertical edge.} \end{cases}$$

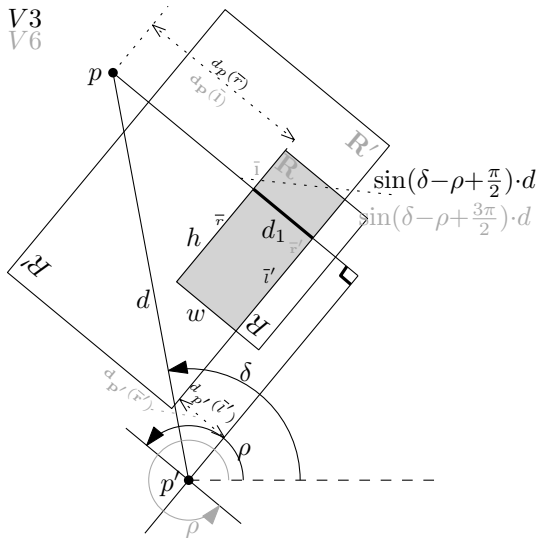
where h_i, v_i, δ and e are chosen as described above.



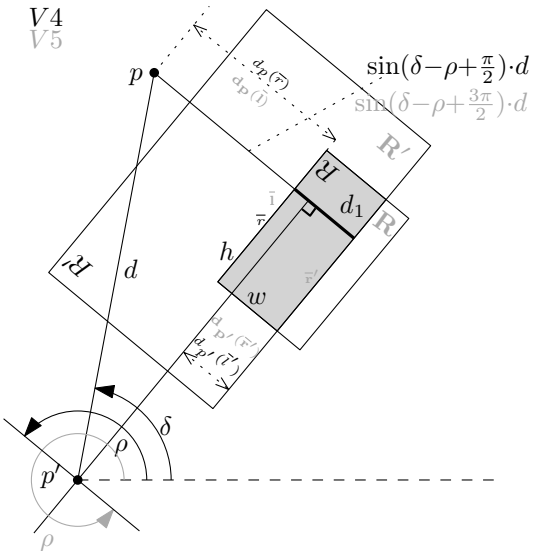
(a) Black: p lies to the **right** of $\overline{r}$ and p' lies to the **right** of $\overline{l}'$. Gray: p lies to the **left** of $\overline{l}$ and p' lies to the **left** of $\overline{r}'$.



(b) Black: p lies to the **right** of $\overline{r}$ and p' lies to the **left** of $\overline{l}'$. Gray: p lies to the **left** of $\overline{l}$ and p' lies to the **right** of $\overline{r}'$.



(c) Black: p lies to the **left** of $\overline{r}$ and p' lies to the **right** of $\overline{l}'$. Gray: p lies to the **right** of $\overline{l}$ and p' lies to the **left** of $\overline{r}'$.



(d) Black: p lies to the **left** of $\overline{r}$ and p' lies to the **left** of $\overline{l}'$. Gray: p lies to the **right** of $\overline{l}$ and p' lies to the **right** of $\overline{r}'$.

Figure 15.19: The rotation phase T2. Shows the case that the edges $\overline{t}$, $\overline{r}$, $\overline{b}$ of R and the edge $\overline{l}'$ of R' enclose the intersection area (black labels) and the case that the edges $\overline{t}$, $\overline{l}$, $\overline{b}$ of R and the edge $\overline{r}'$ of R' enclose the intersection area (gray labels).

Type T2: Assume that two edges e_1 and e_2 of R are involved in the intersection, we then have to distinguish whether both edges have the same or different orientations. If both edges have the same orientation (without loss of generality we assume that they are vertically oriented), then we obtain a situation as depicted in Figure 15.17b. In that case we can describe the intersection as:

$$A_2(R, R', \alpha, \beta) = \int_{\alpha}^{\beta} h' \cdot w \; d\rho = [h' \cdot w \cdot \rho]_{\rho=\alpha}^{\rho=\beta}$$

If both edges are of different orientation, we obtain a situation as depicted in Figure 15.17d. This time we can reuse the results of type T3, because they help us to compute the distances d_v and d_h as depicted in Figure 15.17d. We therefore can use the individual sub-cases $H1 - H8$ and $V1$-$V8$ in order to compute the intersection area between R and R'. For that purpose let $H = \{h_1, \ldots, h_8\}$ and $V = \{v_1, \ldots, v_8\}$ be the constants as defined in the case T2, then for all $h_i \in H$ and all $v_j \in V$ the intersection area is:

$$\begin{aligned}
A_{h_i,v_j,\delta}(R, R', \alpha, \beta) &= \int_{\alpha}^{\beta} (h_i + \operatorname{sgn}(i) \cdot \sin(\rho - \delta)) \cdot (v_j + \operatorname{sgn}(j) \cdot \sin(\rho - \delta - \frac{\pi}{2})) \; d\rho \\
&= \int_{\alpha}^{\beta} h_i \cdot v_j - h_i \cdot \operatorname{sgn}(j) \cdot \cos(\rho - \delta) + v_j \cdot \operatorname{sgn}(i) \cdot \sin(\rho - \delta) \\
&\quad - \operatorname{sgn}(i) \cdot \sin(\rho - \delta) \cdot \operatorname{sgn}(j) \cdot \cos(\rho - \delta) \; d\rho \\
&= [h_i \cdot v_j \cdot \rho]_{\rho=\alpha}^{\rho=\beta} - [h_i \cdot \operatorname{sgn}(j) \cdot \sin(\rho - \delta)]_{\rho=\alpha}^{\rho=\beta} \\
&\quad - [v_j \cdot \operatorname{sgn}(i) \cdot \cos(\rho - \delta)]_{\rho=\alpha}^{\rho=\beta} \\
&\quad + [\frac{1}{2} \operatorname{sgn}(i) \cdot \operatorname{sgn}(j) \cdot (\cos(\rho - \delta))^2]_{\rho=\alpha}^{\rho=\beta}
\end{aligned}$$

Finally, we define

$$A_2(R, R', \alpha, \beta) := A_{h_i,v_j,\delta}(R, R', \alpha, \beta),$$

where h_i, v_j and δ are chosen such that those parameters correspond to the given anchored rectangles R and R'.

15.4.2 Defining the Area Based Visibility of Labels

Based on the previous section we now can introduce an alternative notion of visibility that depends on the area of a label that is visible within the viewport: Assume that we are given a set L of labels and a viewport $\mathcal{VP}$ with trajectory $T = (A_1, \ldots, A_n)$:

For each arc A_i with $1 \leq i \leq n$ we first determine the different phases the label can have regarding $\mathcal{VP}$. To that end we create a sequence

$$P_{A_i}(l) = (([s_1, e_1], t_1, i), \ldots, ([s_k, e_k], t_k, i))$$

such that

- k is minimized, and
- $0 = s_1 \leq e_1 \leq s_2 \leq \ldots \leq e_{k-1} \leq s_k \leq e_k = 1$, and

- for all $1 \leq i \leq k$ and for all $pos \in [s_i, e_i]$ it is true that t_i denotes the number of edges of $\mathcal{VP}$ involved in the intersection of $\mathcal{VP}$ and l at the position pos, and
- for all $pos \in [0, 1]$ there is an element $([s_j, e_j], t_j, i) \in P_{A_i}(l)$ with $pos \in [s_j, e_j]$.

We do not explain explicitly how to gain $P_{A_i}(l)$, but basically one has to translate the start and end angles of the single phases into positions regarding T. This can be done analogous to the approach described in Section 15.3. Afterwards we concatenate those sequences to one long sequence

$$P_T(l) = P_{A_1}(l) + \ldots + P_{A_m}(l)$$

So far we have defined the weight function w of Problem 14.2 that $w(l, s, e) = e - s$ for all labels $l \in L$ and all segments $[s, e] \in S_T^{\mathcal{VP}}(l)$. Now we also want to use the relative intersection area of a label and the viewport. We therefore set:

$$\forall l \in L \; \forall [s, e] \in S_T^{\mathcal{VP}}(l) \colon w(l, s, e) = \frac{A(\mathcal{VP}, l, s, e)}{(e - s) \cdot w_l \cdot h_l}$$

where w_l denotes the width of l and h_l denotes the height of l and where for a label l and an interval $[s, e] \in S_T^{\mathcal{VP}}(l)$ the term $A(\mathcal{VP}, l, s, e)$ is defined as follows:

$$A(\mathcal{VP}, l, s, e) = \sum_{([pos_1, pos_2], t, i) \in P_T(l)} A_t(R(\mathcal{VP}, A_i), l, \alpha_T(\max(s, pos_1)), \alpha_T(\min(e, pos_2))),$$

where $R(\mathcal{VP}, A_i)$ is defined in Chapter 14.

The idea of $A(\mathcal{VP}, l, pos_1, pos_2)$ is, that we consider each phase of intersection between $\mathcal{VP}$ and label l separately. According to the previous section we can then use one of the functions $A_0, \ldots, A_4$ to gain the total area for the interval of that particular phase: As it can be that $s < pos_1$ we have to choose the maximum $\max(s, pos_1)$ for the beginning of A_t. Analogously, we have to choose the minimum $\min(e, pos_2)$ for the end of the of A_t. Finally, α_T converts the given position into the rotation of the trajectory at that position.

As the so introduced weight function measures the relative area of a label that is visible within the viewport, labels that are more contained in the viewport than others are preferred for becoming active.

16. Complexity

In this chapter we discuss several aspects of the complexity of the trajectory based labeling problem (TBLP) as formulated in Problem 14.2. While in the first section we prove that TBLP is $\mathcal{NP}$-complete,in the second section we introduce a conflict graph for TBLP describing conflicts between labels regarding a given viewport with trajectory. We then analyze the size of such a graph.

16.1 Complexity

By means of the next theorem we prove that the trajectory based labeling problem is $\mathcal{NP}$-hard. We use the result that the static label number maximization problem is $\mathcal{NP}$-complete for the one-position model as presented in Chapter 13. Intuitively, a dynamic variant of that problem must also be $\mathcal{NP}$-complete, because an instance of the static case can be seen as an instance of the dynamic case without changes.

In order to give a formal proof, we reduce maximum independent set of rectangles (MISR) as defined in Problem 13.1 on TBLP.

Theorem 16.1. *The labeling problem as described in Problem 14.2 is $\mathcal{NP}$-hard for all three activity models AM1, AM2, AM3.*

Proof. Assume that we are given an instance of MIST, that is, a set $L = \{l_1, \ldots, l_n\}$ of rectangles such that each rectangle is placed horizontally aligned in the plane. Then we want to find a maximum cardinality subset S of L such that the rectangles in S do not overlap. We can formalize this problem as:

We want to find a function $\Psi\colon L \to \{0,1\}$ for all rectangles $l \in L$ such that for all $l, l' \in L$ with $l \neq l'$ the sum $\sum_{l \in L} \Psi(l)$ is maximized and Ψ obeys that if l and l' overlap then $\Psi(l) + \Psi(l') \leq 1$.

By means of an algorithm solving TBLP we can easily find Ψ: We identify the rectangles in L with labels that have arbitrary anchors. Then, we draw a bounding box B around all those labels and extend it by a margin of size $\epsilon > 0$ to all sides (see Figure 16.1). We identify the extended box B with the viewport $\mathcal{VP}$ and define the trajectory T as exactly one vertical line segment of length ϵ that starts in the middle of B. Moreover, we set the weight function to $w(l, s, e) = e - s$.

Applying an optimal algorithm solving Problem 14.2 we maximize the sum:

$$\sum_{l \in L} \sum_{[s,e] \in S_T^{\mathcal{VP}}(l)} \Psi_T^{\mathcal{VP}}([s,e], l) \cdot w(l, s, e) \tag{16.1}$$

Since $\mathcal{VP}$ encloses all labels and a margin of size ϵ, it must be true for all labels $l \in L$ that $\Phi_T^{\mathcal{VP}}(l) = ([0,1])$, that is, each label is visible in $\mathcal{VP}$ for the whole time. Further, if two labels $l, l' \in L$ are in conflict with each other, that conflict c is also visible for the whole time because the labels do not rotate. Thus, in general it is true that:

$$\Phi_T^{\mathcal{VP}}(c) = ([0,1]) \text{ for all conflicts } c \text{ in } C(L)$$

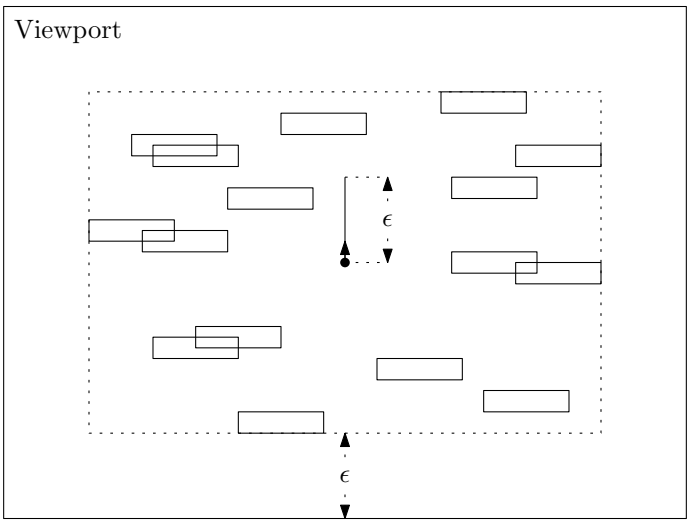


Figure 16.1: Illustration of the proof for Theorem 16.1.

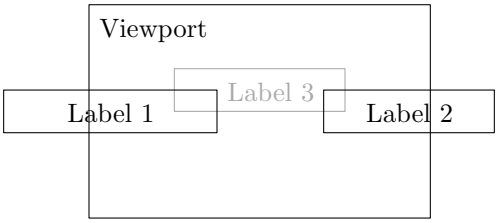


Figure 16.2: Influence of labels. While Label 1 and Label 2 are active, the Label 3 is inactive.

On that account for all three types AM1, AM2 and AM3 a label l can only be active if it wins all its conflicts with other labels. Further, a label that wins all its conflict must be active for the whole time in order to preserve optimality:

$$\text{For all } l \in L\text{:} \sum_{[s,e] \in S_T^{\mathcal{VP}}(l)} \Psi_T^{\mathcal{VP}}([s,e],l) \cdot w(l,s,e) = \Psi_T^{\mathcal{VP}}([0,1],l)$$

Consequently, we can simplify (16.1) to $\sum_{l \in L} \Psi_T^{\mathcal{VP}}([0,1],l)$. Thus, if we set $\Psi_T^{\mathcal{VP}}([0,1],l) = \Psi(l)$ for all labels $l \in L$ we have found an optimal solution for MISR. Obviously, that reduction takes polynomial time. □

16.2 Trajectory Based Conflict Graph

In the next step, we formalize the observation that labels do not necessarily influence the activity of all other labels. For example, assume that we are given only two labels l_1 and l_2 such that they do not collide, then it is obvious that l_1 can not influence the activity of l_2. If we now introduce a third label l_3, that lies in between l_1 and l_2 colliding with both labels, it can be that l_2 influences the activity of l_1: As depicted in Figure 16.2, if we set l_2 active then l_1 also must be active in an optimal solution. Consequently, we obtain a dependency structure that can be modeled as graph:

Definition 16.1 (Conflict Graph)**.** *Given an instance $I = (L, \mathcal{VP}, T)$ of TBLP as defined in Problem 14.2, then we call the multigraph $G = (V, E)$ as defined below the* conflict graph *of I. The nodes of G are defined as:*

$$V = \{\{[s, e], l\} \mid l \in L \text{ and } [s, e] \in \Phi_T^{\mathcal{VP}}(l)\}$$

The edges of G are defined as:

$$E = \{\{\{[s_1, e_1], l\}, \{[s_2, e_2], l'\}\} \mid \{[s_1, e_1], l\} \in V,\ \{[s_2, e_2], l'\} \in V \text{ and there is a conflict } c = (l, l') \in C(L) \text{ such that there is an interval } [s, e] \in \Phi_T^{\mathcal{VP}}(c) \text{ with } [s_1, e_1] \cap [s, e] \cap [s_2, e_2] \neq \emptyset\}.$$

According to that definition we introduce for each label and each visibility of that label exactly one node. Or in other words, each node represents exactly one *visibility* of a label l, which we also call the *label visibility*.

The edges of G then model the conflicts between those label visibilities, that is, we connect two nodes $\{[s_1, e_1], l\}$ and $\{[s_2, e_2], l'\}$ with each other if there is a conflict c between both labels l and l' such that the conflict is visible within $[s_1, e_1]$ and $[s_2, e_2]$. As c can be visible several times within the same visibility of l and l' we obtain a multigraph.

We also write $(c, [s, e], \{[s_1, e_1], l\}, \{[s_2, e_2], l'\}) \in E$ in order to indicate that we refer to the edge $\{\{[s_1, e_1], l\}, \{[s_2, e_2], l'\}\} \in E$ that is induced by the conflict c that is visible within the interval $[s, e]$ such that $[s_1, e_1] \cap [s, e] \cap [s_2, e_2] \neq \emptyset$. On that account, we also say that an edge $f \in E$ is a conflict between two visibilities in V and call f the *conflict visibility*.

The next lemma makes a statement about the size of V and E:

Lemma 16.1. *Given an instance $I = (L = \{l_1, \ldots, l_n\}, \mathcal{VP}, T = (A_1, \ldots, A_m))$ TBLP as defined in Problem 14.2 and let $G = (V, E)$ be the corresponding conflict graph, then it is true that $|V| \in \mathcal{O}(n \cdot m)$ and $|E| \in \mathcal{O}(n^2 \cdot m^2)$.*

Proof. We first show that $|V| \in \mathcal{O}(n \cdot m)$: From Lemma 15.2 we know that each arc A_i in T can induce at most five visibilities for a label $l \in L$. Thus, for T we create at most $\mathcal{O}(m)$ visibilities for l so that the claim follows.

In the worst case a visibility $v \in V$ of a label l' can be in conflict with all visibilities $v' \in V$ of all other labels $l' \in L$ with $l' \neq l$ so that we obtain the claim $|E| \in \mathcal{O}(n^2 \cdot m^2)$. □

In many practical cases the size of G does not match its worst-case bounds, but G decomposes into components of a graph separately. For solving TBLP we can apply the same strategy on all individual components without losing optimality, because label visibilities of different components cannot influence each other. Hence, we therefore assume in the further parts without loss of generality that the considered conflict graphs are connected.

17. Algorithmic Approaches

In this chapter we discuss two different kinds of algorithmic approaches for solving TBLP: In the first section we present for the different activity models of TBLP corresponding formulations using integer linear programming, such that we can solve TBLP optimally. In the second section we then introduce simple heuristics for the activity model AM1.

17.1 Integer Linear Programming

In this section we discuss for each type of activity model separately how TBLP can be modeled using integer linear programming. The main idea is to translate the conditions for TBLP as formulated in Problem 14.2 into linear equations and inequations subject to which one tries to maximize a objective function measuring the overall activity of all labels.

We first introduce an ILP for AM1, then for AM3 and finally based on the ILP for AM3 an ILP for AM2. For all three cases we assume that we are given a set $L = \{l_1, \ldots, l_n\}$ of labels, a viewport $\mathcal{VP}$ with trajectory $T = (A_1, \ldots, A_m)$ and a weight function $w\colon L \times [0,1] \times [0,1] \to \mathbb{R}$ for the visibility of labels. Further, we assume that $G = (V, E)$ is the corresponding conflict graph of that instance. As already mentioned in Section 16.1 we assume without loss of generality that G is connected, because otherwise we can separately apply the upcoming approach on the connected components of G.

In the following we say for a conflict visibility $f{=}(c, [s, e], v, v') \in E$ that v *wins* f if the corresponding label of v is set active within $[s, e]$. Analogously, we say v' *wins* f if the corresponding label of v' is set active within $[s, e]$. Among other things, for the upcoming ILPs we need to introduce constraints that enforce that a conflict visibility can only be won by one of both label visibilities.

17.1.1 ILP for AM1

The idea of the activity model AM1 is that labels may only become active when they become visible within the given viewport and they can only become inactive when they become invisible. In other words, we can say that a label l becomes active for a visibility $[s, e] \in \Phi_T^{\mathcal{VP}}(l)$ if and only if that visibility wins all conflicts of l that are visible within $[s, e]$. We therefore have to find a maximum weighted independent set in G (for more information about maximum weighted independent sets see for example [KOHH05]):

Variables: For each conflict visibility $f{=}(c, [s, e], v, v') \in E$ we introduce two variables $z_f^{v,v'} \in \{0, 1\}$ and $z_f^{v',v} \in \{0, 1\}$. The idea is that $z_f^{v,v'} = 1$ ($z_f^{v',v} = 1$) if and only if v (v') wins the visibility f of c. We say that $z_f^{v,v'}$ *belongs to* v and that $z_f^{v',v}$ *belongs to* v'. We denote the set of all variables that belong to v by:

$$Z_v{:=}\{z_f^{v,v'} \mid \{v, v'\} \in E\}$$

Constraints: In order to guarantee that a conflict visibility $f{=}(c,[s,e],v,v') \in E$ of a conflict can be won by one of both label visibilities v and v', we introduce for all conflict visibilities $f{=}(c,[s,e],v,v') \in E$ the condition:

$$z_f^{v,v'} + z_f^{v',v} \leq 1 \tag{17.1}$$

Further, we want each label visibility $v \in V$ to win either all or none of its conflicts. Thus, we introduce for the set $Z_v = \{z_f^{v,v_1}, \ldots, z_f^{v,v_k}\}$ the following condition:

$$z_f^{v,v_1} = \ldots = z_f^{v,v_k} \tag{17.2}$$

Objective Function: Since we want to maximize the overall activity over all labels we maximize the following function:

$$\sum_{v=\{[s,e],l\} \in V} rep(Z_v) \cdot w(l,s,e)$$

where $rep(Z_v)$ denotes one arbitrarily chosen representative variable of Z_v.

Interpretation: After solving that ILP we set for each label visibility $v{=}\{l,[s,e]\} \in V$:

$$\Psi_T^{\mathcal{VP}}([s,e],l) = rep(Z_v)$$

Complexity: In order to obtain rough indications about the complexity of the ILP, we estimate the number of variables and the number of constraints: For each conflict visibility $f \in E$ we introduce exactly two variables, so that we obtain $\mathcal{O}(n^2 \cdot m^2)$ variables by means of Lemma 16.1. Further, for each conflict visibility $f \in E$ we introduce the Constraint (17.1) and for each label visibility $v \in V$ we introduce the Constraint (17.1), which consists of $\mathcal{O}(n{\cdot}m)$ sub-constraints. Consequently, we obtain $\mathcal{O}(n^2 + m^2)$ constraints.

17.1.2 ILP for AM3

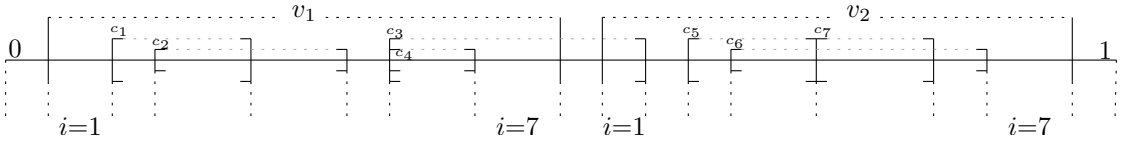
As AM2 is a direct specialization of AM3 we first explain the ILP for AM3. AM3 is characterized by the requirement that a label l may become active, when there is a witness l' for setting l active (see Definition 14.3 and Definition 14.4). Analogously, a label l may become inactive when there is a witness l' for setting l inactive.

In order to satisfy that requirement we pursue the idea that for a label l we can divide T into atomic segments $S_T^{\mathcal{VP}}(l)$ (see Definition 14.5, page 131), such that each optimal activity of l is composed of those segments. In particular we consider for each label visibility $v{=}\{l,[s,e]\} \in V$ of l the corresponding segments on T:

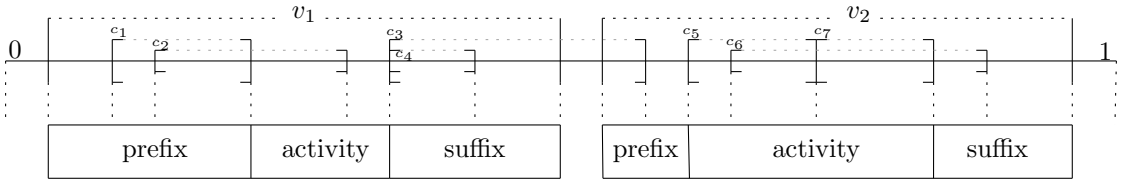
$$S_T^{\mathcal{VP}}(l,[s,e]) = \{([pos_1,pos_2],i) \mid [pos_1,pos_2] \text{ is the } i\text{-th segment in } S_T^{\mathcal{VP}}(l) \text{ with } [pos_1,pos_2] \cap [s,e] \neq \emptyset\}$$

For a label visibility $v{=}\{l,[s,e]\} \in V$ we also write $S_T^{\mathcal{VP}}(v)$ instead of $S_T^{\mathcal{VP}}(l,[s,e])$. Further, note that the counter variable i starts for each label visibility $v \in V$ with 1 (see Figure 17.1a).

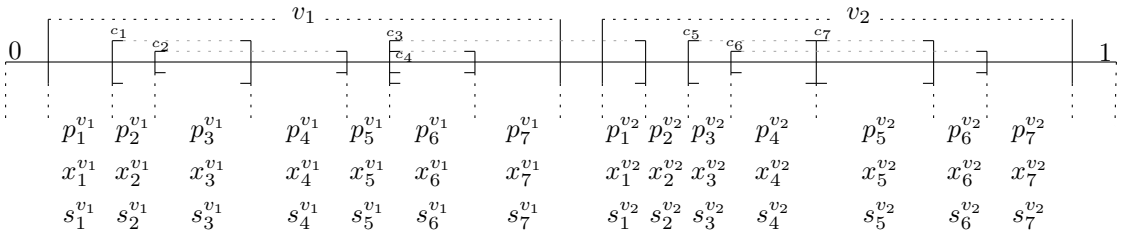
By means of those segments $S_T^{\mathcal{VP}}(v)$ we can divide each label visibility $v \in V$ into three blocks (see Figure 17.1b): an inactive prefix block, an active block and finally an inactive suffix block. We then use for the ILP the simple property that each segment of $S_T^{\mathcal{VP}}(l,[s,e])$ must belong to exactly one of those three blocks.



(a) Each label visibility is divided into segments.



(b) Each label visibility is described by three blocks.



(c) The variables P_{v_1}, X_{v_1} and S_{v_1} for v_1 and the variables P_{v_2}, X_{v_2} and S_{v_2} for v_2.

Figure 17.1: Segmentation of label visibilities. For a label l with two visibilities v_1 and v_2 different views on the segmentation of T are illustrated.

Variables: Analogously to AM1 we introduce for each label visibility $f{=}\{c, [s,e], v, v'\} \in E$ of a conflict two variables $z_f^{v,v'} \in \{0,1\}$ and $z_f^{v',v} \in \{0,1\}$ modeling the visibility of conflicts. We again define

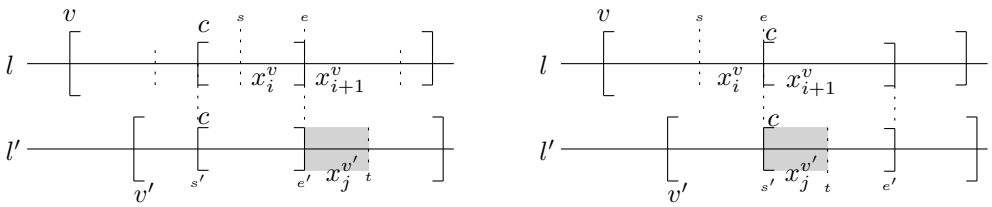
$$Z_v{:=}\{z_f^{v,v'} \mid \{[s,e], v, v'\} \in E\}$$

In order to model the prefix-, activity- and suffix-block for the ILP we introduce for each segment $([pos_1, pos_2], i) \in S_T^{\mathcal{VP}}(v)$ of each label visibility $v \in V$ the following variables:

1. $p_i^v \in \{0,1\}$: $p_i^v = 1$ if and only if its belongs to the prefix of v.
2. $x_i^v \in \{0,1\}$: $x_i^v = 1$ if and only if its belongs to the activity of v.
3. $s_i^v \in \{0,1\}$: $s_i^v = 1$ if and only if its belongs to the suffix of v.

Figure 17.1c illustrates how the variables are distributed over the trajectory T regarding one label l. Further, we define the sets containing for each label visibility v all variables of a certain kind of block:

1. $P_v = \{p_i^v \mid ([pos_1, pos_2], i) \in S_T^{\mathcal{VP}}(v)\}$
2. $X_v = \{x_i^v \mid ([pos_1, pos_2], i) \in S_T^{\mathcal{VP}}(v)\}$
3. $S_v = \{s_i^v \mid ([pos_1, pos_2], i) \in S_T^{\mathcal{VP}}(v)\}$



(a) Segment j is witness for setting l active after segment i: The conflict c of l and l' ends at the beginning of j.

(b) Segment j is witness for setting l inactive after segment i: The conflict c of l and l' begins at the beginning of j.

Figure 17.2: The concept of witnesses for conflicts. For two labels l and l' with visibilities v' and v' and common conflict c the sets WA_v^i and WI_v^i are illustrated. Dotted lines indicate the transition of two segments and the gray segment indicates the witness for setting l active/inactive after the segment corresponding to x_i^v.

Constraints: First we introduce a constraint stating that each segment $([pos_1, pos_2], i) \in S_T^{\mathcal{VP}}(v)$ of each label visibility $v \in V$ belongs to exactly one block:

$$p_i^v + x_i^v + s_i^v = 1 \tag{17.3}$$

Further, we have to ensure that those variables actually describe blocks. Hence, we introduce for each label visibility $v \in V$ with prefix variables $P_v = \{p_1^v, \ldots, p_k^v\}$ and suffix variables $S_v = \{s_1^v, \ldots, s_k^v\}$ the following two conditions:

$$p_1^v \geq \ldots \geq p_k^v \tag{17.4}$$

$$s_1^v \leq \ldots \leq s_k^v \tag{17.5}$$

In the next step we enforce that a label l can only become active when it becomes visible or there is a witness l' for setting l active. Analogously, we have to enforce that a label l can only become inactive when it becomes invisible or there is a witness l' for setting l inactive.

In order to solve that problem we define for each label visibility $v \in V$ and for each segment $([s, e], i) \in S_T^{\mathcal{VP}}(v)$ the set

$$\begin{aligned}\text{WA}_v^i = \{x_j^{v'} \in X_{v'} \mid & v' \in V \text{ and } ([e', t], j) \in S_T^{\mathcal{VP}}(v') \text{ such that there is a conflict visibility} \\ & (c, [s', e'], v, v') \in E \text{ with } e' = e \text{ and the conflict } c \text{ ends at } e'.\}\end{aligned}$$

that contains all witnesses for v testifying that l may become active after the given segment $([s, e], i)$. Figure 17.2a illustrates the set for a certain i.

Analogously, we define for each label visibility $v \in V$ and for each segment $([s, e], i) \in S_T^{\mathcal{VP}}(v)$ the set

$$\begin{aligned}\text{WI}_v^i = \{x_j^{v'} \in X_{v'} \mid & v' \in V \text{ and } ([s', t], j) \in S_T^{\mathcal{VP}}(v') \text{ such that there is a conflict visibility} \\ & \{c, [s', e'], v, v'\} \in E \text{ with } s' = e \text{ and the conflict } c \text{ begins at } s'.\}\end{aligned}$$

that contains all witnesses for v testifying that l may become inactive after the given segment $([s, e], i)$. Figure 17.2b illustrates the set for a certain i.

Then we introduce for each label visibility $v \in V$ and for all consecutive activity variables $x_i^v, x_{i+1}^v \in X_v$ $(1 \leq i < k)$ with witnesses $\text{WA}_v^i = \{x_1, \ldots, x_t\}$ the following condition:

$$x_{i+1}^v - x_i^v - x_1 - \ldots - x_t \leq 0 \tag{17.6}$$

The following table helps to understand that condition:

x_i^v	x_{i+1}^v	$x_{i+1}^v - x_i^v$
0	0	0
0	1	1
1	0	−1
1	1	0

Obviously, apart from the second row for all other rows the Constraint (17.6) is true. The second row describes the case that the i-th segment of v is inactive while the next segment is active. For that case the Constraint (17.6) becomes only true if there is at least one witness in WA_v^i that is also set to 1. Consequently, the condition enforces that a label l can only become active between two consecutive segments if l becomes visible or there is a witness l' for setting l active.

Analogously, we introduce for all consecutive variables $x_i^v, x_{i+1}^v \in X_v$ $(1 \leq i < k)$ with witnesses $\text{WI}_v^i = \{x_1, \ldots, x_t\}$ the following condition:

$$x_i^v - x_{i+1}^v - x_1 - \ldots - x_t \leq 0 \tag{17.7}$$

Then we can argue for the soundness of that condition by means of the following table:

x_i^v	x_{i+1}^v	$x_i^v - x_{i+1}^v$
0	0	0
0	1	−1
1	0	1
1	1	0

Apart from the third row for all other rows the Constraint (17.7) is true. The third row describes the change from activity to inactivity from the i-th segment of v to the $(i+1)$-th segment of v. For that case the Constraint (17.7) becomes only true if there is at least one witness in WI_v^i that is also set to 1. Consequently, a label l can only become inactive between two segments if l becomes invisible or there is witness l' for setting l inactive.

So far we have not considered conflicts between labels yet: In order to guarantee that a conflict visibility $f{=}(c, [s, e], v, v') \in E$ can be won only by one of both label visibilities v and v', we introduce for all visibilities $f{=}(c, [s, e], v, v') \in E$ the condition:

$$z_f^{v,v'} + z_f^{v',v} \leq 1 \tag{17.8}$$

Then we also need to enforce that a label can only be active at a certain segment, if it has won all conflicts that overlap that segment: We therefore introduce for all visibilities $v \in V$ and all segments $([pos_1, pos_2], i) \in S_T^{\mathcal{VP}}(v)$ and all conflicts $f{=}(c, [s', e'], v, v') \in E$ with $[s', e'] \cap [pos_1, pos_2] \neq \emptyset$:

$$x_i^v \leq z_f^{v,v'} \tag{17.9}$$

Consequently, two labels cannot be active at the same position, if they have a conflict at that particular position.

Objective Function: For each visibility of a label we collect all segments that are set to be active and maximize the sum

$$\sum_{v \in V} \sum_{\substack{([p_1,p_2],i) \in \\ S_T^{\mathcal{VP}}(v)}} x_i^v \cdot w(l, p_1, p_2)$$

Interpretation: For each label visibility $v \in V$ and each segment $([p_1, p_2], i) \in S_T^{\mathcal{VP}}(v)$ we set:

$$\Psi_T^{\mathcal{VP}}([p_1, p_2], l) = x_i^v$$

Complexity: Analogously to ILP1, we introduce for each conflict visibility $f=\{v, v'\} \in E$ the two variables $z_f^{v,v'}$ and $z_f^{v',v}$ so that we obtain $\mathcal{O}(n^2 \cdot m^2)$ variables for all conflicts by means of Lemma 16.1. In order to estimate the prefix-, activity- and suffix-variables, we consider the set $S_T^{\mathcal{VP}}(v)$ for a label visibility $v \in V$: As every label visibility $v \in V$ has $\mathcal{O}(m \cdot n)$ conflicts (Lemma 16.1), the set $S_T^{\mathcal{VP}}(v)$ contains $\mathcal{O}(m \cdot n)$ elements. Thus, we introduce $\mathcal{O}(m^2 \cdot n^2)$ variables describing blocks. On that account, we use $\mathcal{O}(n^2 \cdot m^2)$ variables.

It is easy to see that we obtain $\mathcal{O}(n^2 \cdot m^2)$ constraints for (17.3),(17.4),(17.5) and (17.8). Further, in the worst case we introduce for each label visibility $v \in V$ and each segment of $S_T^{\mathcal{VP}}(v)$ one instance of Constraint (17.6) and one instance of Constraint (17.7). Thus, we obtain $\mathcal{O}(n^2 \cdot m^2)$ constraints for (17.6) and (17.7).

For Constraint (17.9) we consider how many segments of a label visibility v a conflict $f \in E$ can overlap: As $S_T^{\mathcal{VP}}(v)$ contains $\mathcal{O}(m \cdot n)$ elements, each conflict of v can overlap $\mathcal{O}(m \cdot n)$ segments. Further, as v has $\mathcal{O}(m \cdot n)$ conflicts, we introduce $\mathcal{O}(m^2 \cdot n^2)$ instances of constraint (17.9) for v. Thus, we obtain $\mathcal{O}(m^3 \cdot n^3)$ instances of Constraint (17.9) in total.

On that account we obtain $\mathcal{O}(m^2 \cdot n^2)$ variables and $\mathcal{O}(m^3 \cdot n^3)$ constraints.

17.1.3 ILP for AM2

AM2 is characterized by the requirement that a label l can only become active when it becomes visible, while it can become inactive when it becomes invisible or there is witness l' for setting l inactive. In other words, prefixes are always empty.

Based on the ILP for AM3 we can easily introduce an ILP for AM2. We only have to introduce for each visibility $v \in V$ with prefix variables $P_v = (p_1^v, \ldots, p_k^v)$ the condition:

$$p_1^v = \ldots = p_k^v = 0 \tag{17.10}$$

Due to that condition we know that the prefix always has a length of 0. Practically, it is more efficient to not introduce the prefix at all, that is, the visibility $v \in V$ is only divided into two blocks. In particular then the Constraint (17.3) changes to

$$x_i^v + s_i^v = 1 \tag{17.11}$$

Obviously, the complexity of that ILP does not change. Note that we cannot use that approach in order to obtain an ILP for AM1, because if we also set $s_1^v = \ldots = s_k^v = 0$ for all suffix variables $S_v = (s_1^v, \ldots, s_k^v)$ of all visibilities v, then by Constraint (17.3) we can conclude that $x_1^v = \ldots = x_k^v = 1$ for all activity variables $X_v = (x_1^v, \ldots, x_k^v)$ of all visibilities v.

17.2 The Heuristics

We shortly explain what kind of heuristics we have implemented. For that purpose assume that we are given a map M with labels $L = \{l_1, \ldots, l_n\}$ and a viewport $\mathcal{VP}$ with trajectory $T = \{A_1, \ldots, A_m\}$. Further, let $G = (V, E)$ be the conflict graph as introduced in Definition 16.1. The presented heuristics work on G and provide only the activity model AM1 for labels (see Definition 14.4). Further, all of the upcoming heuristics use the same framework:

1. Rank all label visibilities in V by means of a weight function.
2. Choose the visibility $v=\{l,[s,e]\} \in V$ of highest rank.
3. Delete v and all its neighbors $v' \in V$ (there is an edge $\{c,[s',e'],v,v'\} \in E$).
4. Set the label l for the interval $[s,e]$ active.
5. (Update the ranking of the resulting visibilities.)
6. If V is not empty, repeat steps 2 - 6.

Step 5 is optional. In the case that we make use of that step we call the heuristic *dynamic* and otherwise *static*. We have considered the following heuristics, which differ by the choice of the ranking function.

Greatest Visibility (GV): As the name already suggests we rank the visibilities $v \in V$ by their length and choose the label with greatest visibility as the next label to be set active. Since deleting v from V does not influence the remaining visibilities in V, that heuristic is static. Obviously, the time consuming part of that heuristic is sorting the visibilities by rank. Consequently, we need $\mathcal{O}(|V| \log |V|)$ time. From Lemma 16.1 we know that $|V| \in \mathcal{O}(n \cdot m)$, so that we gain $\mathcal{O}(n \cdot m \cdot \log(m \cdot n))$ time.

Least Influence (LI): This time we do not consider the length of the visibilities V, but we measure the influence they have: To that end we determine for each visibility $v \in V$ the following sum:

$$\text{influence(v)} = \sum_{\substack{v'=(l',[s,e]) \in V \\ v' \text{ is neighbor of } v}} (e-s)$$

Then we rank the visibilities in V such that a visibility v with small influence has a high priority. If we do not update the influence of the remaining visibilities, we obtain a static heuristic that we abbreviate by LIS. In the case that we update the influence of the remaining visibilities, we obtain a dynamic heuristic, which we abbreviate by LID.

For the static case we need at most $\mathcal{O}(n^2 \cdot m^2)$: From Lemma 16.1 we know that there are at most $\mathcal{O}(n \cdot m)$ visibilities and for each visibility there are at most $\mathcal{O}(n \cdot m)$ conflicts. Even though the result seems to be not very good, the worst case is very unrealistic for an appropriate setting: Normally, the map is huge in comparison to the size of the viewport, so that not every label is visible for the whole time. On that account the running time for this heuristic is normally much better than the worst case indicates (see Section 18.2).

Fewest Conflicts (FC): We rank that visibility $v \in V$ highest that has the fewest conflicts with other visibilities, or in other words it has fewest edges in E. In the case that we do not update the ranking after each deletion step, we abbreviate the heuristic by FCS and in the other case by FCD. Again, the conflict graph can contain $\mathcal{O}(n \cdot m)$ visibilities and each visibility can have at most $\mathcal{O}(n \cdot m)$, so that we need at most $\mathcal{O}(n^2 \cdot m^2)$ time. But as already mentioned in the previous section for real world data the worst case is very unlikely for appropriate zoom levels: Normally, a visibility has only a low number of conflicts (see Section 18.2).

18. Experimental Evaluation

In this chapter we present some experimental evaluation of the implementation of the previous chapters. To that end we have also implemented the ILPs and heuristics as presented in Chapter 17 in order to evaluate those on real-world data. While in the first section we describe which data and which parameters we have chosen, in the second section we present the actual results and discuss them.

18.1 Setting

In this section we explain how we have processed the evaluation. In particular we explain which data we have used and how the parameters have been chosen.

Basically, we need labels and trajectories for the evaluation. We have decided to do the evaluation on real world data, so that one gets an idea how well the approaches work. To that end we have extracted the labels from the digital map OpenStreetMap[1], which is known for its high level of detail: As many people contribute to that project based on the wikipedia principle the map is well elaborated for many regions.

The labels that can be extracted are of different categories as *highway*, *shops* and *tourism*. In order to get an realistic view with appropriate many labels, we have chosen the category *amenity* which comprises a high number of labels:

> *"Covering a wide variety of civil amenities including car parks, hospitals, schools, bus stations and libraries."*[2]

We have chosen Karlsruhe to be the region for the evaluation and distinguish two datasets: One that embraces the city of Karlsruhe and one that also embraces the countryside of Karlsruhe (see Table 18.1). We distinguish these two data sets because the density of labels is significantly different for cities and countrysides.

For computing the trajectories we have used a road graph that corresponds to the region of the labels. Basically, we have computed shortest paths between 1000 randomly chosen pairs

Property	Value
Size	179 $km \times 150$ km
Number of Labels	26274
Label Density	0.97 $\frac{\text{Labels}}{km^2}$
Latitude	$48.323 - 49.670$
Longitude	$7.961 - 9.570$

(a) Countryside of Karlsruhe

Property	Value
Size	21 $km \times 5$ km
Number of Labels	2304
Label Density	21.9 $\frac{\text{Labels}}{km^2}$
Latitude	$48.994 - 49.041$
Longitude	$8.312 - 8.503$

(b) City Karlsruhe

Table 18.1: Key data for the considered maps.

[1] http://www.openstreetmap.org/

[2] http://wiki.openstreetmap.org/wiki/Amenities

Property	Value
Max. Number of Arcs	1900
Average Number of Arcs	776.3
Max. Length	216.4 km
Average Length	90.6 km

(a) Countryside of Karlsruhe

Property	Value
Max. Number of Arcs	282
Average Number of Arcs	96.1
Max. Length	21.7 km
Average Length	7.5 km

(b) City Karlsruhe

Table 18.2: Key data for the computed trajectories.

of street nodes. Afterwards, we simplified the polygonal path by smooth polyarcs as described in Section 6.1. Table 18.2 shows some key data for those paths.

After we have discussed which data we use for the evaluation, we now explain how we define the viewport size and the area that can be seen within the viewport. As navigation systems are the major use case that we consider, we have decided to set the size of the viewport to the resolution of a customary navigation system, namely to the resolution of 480×272 pixels. Further, we apply the evaluation twice, once for a scale of $1 : 2000$ and once for a scale of $1 : 4000$. For the former case this means that a region of $487\ m \times 397\ m$ is visible within the viewport and for the latter case a region of $974\ m \times 782\ m$ is visible.

We process the evaluation as follows: For both given datasets we apply both the ILPs and the heuristics in order to compute the activity of the labels. As already mentioned, we do this twice: once for a scale of $1 : 2000$ and then for a scale of $1 : 4000$. To that end we used the programming language Java for the implementation and applied the evaluation on a system with 2GB RAM and Intel®Core™2 Duo CPU 2.00GHz×2 using Ubuntu as operating system. Only for the ILPs concurrency is applied.

18.2 Results of the Evaluation

For each considered trajectory we have computed two kinds of data: On the one hand we have analyzed the conflict graph (see Definition 16.1) of that trajectory and on the other hand we have computed different activities $\Psi_T^{\mathcal{VP}}$ for the corresponding labels.

Visibility Graph: For the conflict graph $G = (V, E)$ of a viewport $\mathcal{VP}$ with trajectory T we have considered the following parameters:

1. Visibilities: Corresponds to the number of nodes within V and states how many visibilities of labels exist in total.

2. Max. Degree: The maximal degree within G, that is, the maximal conflict count of a visibility $v \in V$.

3. Avg. Degree: The average degree within G, that is, the average conflict count of a visibility $v \in V$.

4. Components: The number of components the graph G consists of.

5. $\frac{\text{Max. Visibilities}}{\text{Component}}$: Maximal number of nodes per components, that is, the maximal number of visibilities per component.

6. $\frac{\text{Avg. Visibilities}}{\text{Component}}$: Average number of nodes per components, that is, the average number of visibilities per component.

As we consider 1000 trajectories per data set, we do not present all data that we have collected, but only the average, minimum and maximum of that data over all 1000 trajectories. The data for the conflict graphs of the different data sets are presented in Table 18.3-18.6.

It is remarkable that the conflict graphs of all data sets decompose into a high number of components regarding the number of visibilities. Consequently each component of a conflict graph consists only of few visibilities so that there is the hope that the problem remains solvable in practice, even though the considered labeling problem is $\mathcal{NP}$-complete. Further, the high number of components lends itself to use concurrent approaches that work on the sub-problems independently from each other.

As is to be expected, the zoom level has a great influence on the parameters. For the scale factor 1:4000 the number of visibilities and the conflict count increase regarding the scale factor 1:2000. It can be simply explained by the fact that for a scale factor 1:4000 the labels are more dense then for a scale factor 1:2000.

As the data set Karlsruhe Countryside subsumes the data set Karlsruhe City, mainly the average values for that data set are interesting: The average degree of the conflict graphs for Karlsruhe Countryside is a bit less than the average degree of the conflict graphs for Karlsruhe City, which can be explained by the fact that the density of labels per km^2 is significantly smaller for the former one than for the latter one (see Table 18.1). The number of components cannot directly be compared, because the length of the considered trajectories is much longer for Karlsruhe Countryside than for Karlsruhe City (see Table 18.2).

Activity: For each trajectory we have computed possible activities using the ILPs as presented in Section 17.1 and the heuristics as presented in Section 17.2: Table 18.7-18.10. We compare those activities relative to the case that we do not resolve the occurring conflicts, that is, the case 'Without Resolving' has an activity of 100 %, while the other strategies have only a fraction of that activity. By doing so, we can compute the results of the given ILPs and heuristics pairwise.

For each considered data set it is true that the difference between the single ILPs lies below 5.9% so that ILP2 and ILP3 can be seen as a kind of fine tuning for ILP1.

Surprisingly, the difference between the heuristics of a single data set is small, namely below 0.9% in the average case. In comparison to the given ILPs the quality of the heuristics is mainly based on the used scale factor. As is to be expected for the scale factor 1:2000 we obtain better results than for the scale factor 1:4000: While for the former scale factor the heuristic yield similar results as the optimal ILPs, for the latter scale factor the difference between heuristics and ILPs becomes magnificent.

The results for the data sets Karlsruhe City and Karlsruhe Countryside are in the same proportion good: We cannot ascertain a significant difference between both types of data sets, but only a little difference in favor of Karlsruhe Countryside. We assume that one can explain that difference by means of the lower density of labels that Karlsruhe Countryside possesses (see Table 18.1).

Time Consumption: In Table 18.11-18.12 we present the times that were necessary for computing the conflict graph and applying the ILPs and heuristics on the different data sets. Again, we measured for the given trajectories the individual times and present the average and maximum of these times. As we also considered short trajectories, we obtained for all

combinations of data sets and parameters a minimum time less than one millisecond. We therefore do not explicitly state the minimum time within the tables.

Further, the times for creating the conflict graphs only serve as rough indications for realistic running times: So far we have not optimized the computation of a conflict graph, that is, for each arc of the trajectory we still consider each label when computing the visibilities. Usually, the labels are spread over the map so that in the practical use case we do not need to consider each label for each arc, but we can apply an approach similar to that one presented in Implementation Details 15.1. On that account, it is likely that conflict graphs can be created much faster. We therefore only present the values in the corresponding tables and resign to discuss them explicitly.

We now discuss the results for the heuristics and the ILPs: As is to be expected applying the dynamic heuristics takes more time than applying static heuristics. Especially, for a scale of 1:4000 the dynamic heuristics are much slower than the static ones: In the average case we obtain a factor of at least 25 for Karlsruhe City and a factor of at least 35 for Karlsruhe Countryside. For the maximum case we obtain even worse factors. As the activity for all heuristics are very similar (see previous paragraph), the static heuristics lend themselves to be the choice.

For ILP1 and ILP2 we obtain running times that mostly lie under 100 milliseconds in average: Only for Karlsruhe Countryside with scale 1:4000 the approach ILP2 takes 151 milliseconds in average and 1.1 second maximal. But as the maximum time of both ILPs is at least 118 milliseconds or worse, it is questionable whether the ILPs can be used for practical purposes. Especially, in the case that one wants to process many requests, as it occurs in a server-client-scenario, short and stable running times are required.

ILP3 is significantly slower than ILP1 and ILP2: For the scale of 1:4000 it can occur that ILP3 needs several minutes to compute the optimal solution. Intuitively, this can be explained by the fact that the activity model AM3 is more complicated than AM1 and AM2. While for ILP1 and ILP2 it is questionable whether they can be used in practice, the approach ILP3 seems only to be feasible for purposes of analysis.

Summarizing the results, we can state that the heuristics can especially be used for small scaled maps, when the running time is essential. If the quality is more important than the running time then ILPs are feasible for gaining optimal solutions.

	Visibilities	Max. Degree	Avg. Degree	Components	$\frac{\text{Max. Visibilities}}{\text{Component}}$	$\frac{\text{Avg. Visibilities}}{\text{Component}}$
Average	177.1	5.2	0.6	133.2	8.0	1.3
Minimum	1.0	0.0	0.0	2.0	1.0	0.5
Maximum	658.0	31.0	2.3	483.0	77.0	2.2

Table 18.3: Data for conflict graphs based on the data set Karlsruhe City 1:2000.

	Visibilities	Max. Degree	Avg. Degree	Components	$\frac{\text{Max. Visibilities}}{\text{Component}}$	$\frac{\text{Avg. Visibilities}}{\text{Component}}$
Average	373.6	13.5	1.7	187.5	33.7	1.9
Minimum	1.0	0.0	0.0	2.0	1.0	0.5
Maximum	1126.0	68.0	4.9	502.0	261.0	4.6

Table 18.4: Data for conflict graphs based on the data set Karlsruhe City 1:4000.

	Visibilities	Max. Degree	Avg. Degree	Components	$\frac{\text{Max. Visibilities}}{\text{Component}}$	$\frac{\text{Avg. Visibilities}}{\text{Component}}$
Average	239.6	6.2	0.5	190.9	8.5	1.2
Minimum	1.0	0.0	0.0	2.0	1.0	0.5
Maximum	1032.0	22.0	2.2	685.0	77.0	2.1

Table 18.5: Data for conflict graphs based on the data set Karlsruhe Countryside 1:2000.

	Visibilities	Max. Degree	Avg. Degree	Components	$\frac{\text{Max. Visibilities}}{\text{Component}}$	$\frac{\text{Avg. Visibilities}}{\text{Component}}$
Average	474.8	16.9	1.5	278.0	43.6	1.6
Minimum	1.0	0.0	0.0	2.0	1.0	0.5
Maximum	1539.0	50.0	6.9	809.0	223.0	2.6

Table 18.6: Data for conflict graphs based on the data set Karlsruhe Countryside 1:4000.

	Without Resolving	ILP1	ILP2	ILP3	GV	LIS	LID	FCS	FCD
Average	100.0	83.4	85.1	86.5	79.2	79.9	79.9	79.6	79.6
Minimum	100.0	56.2	60.8	65.0	41.8	42.1	42.4	40.9	41.5
Maximum	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 18.7: Data for the activity of labels in percent based on the data set Karlsruhe City 1:2000.

	Without Resolving	ILP1	ILP2	ILP3	GV	LIS	LID	FCS	FCD
Average	100.0	67.6	70.5	72.8	58.3	59.2	59.2	58.9	58.9
Minimum	100.0	41.4	47.3	53.1	21.1	21.2	21.0	20.3	21.4
Maximum	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 18.8: Data for the activity of labels in percent based on the data set Karlsruhe City 1:4000.

	Without Resolving	ILP1	ILP2	ILP3	GV	LIS	LID	FCS	FCD
Average	100.0	85.8	87.9	89.5	82.4	82.9	82.9	82.7	82.8
Minimum	100.0	56.6	70.9	73.6	34.4	40.6	40.6	40.6	40.6
Maximum	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 18.9: Data for the activity of labels in percent based on the data set Karlsruhe Countryside 1:2000.

	Without Resolving	ILP1	ILP2	ILP3	GV	LIS	LID	FCS	FCD
Average	100.0	73.5	76.6	79.4	65.1	65.9	65.8	65.6	65.7
Minimum	100.0	42.3	44.4	53.0	21.5	21.5	21.5	21.5	21.5
Maximum	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 18.10: Data for the activity of labels in percent based on the data set Karlsruhe Countryside 1:4000.

	Conflict Graph	ILP1	ILP2	ILP3	GV	LIS	LID	FCS	FCD
Average	0.12	0.017	0.025	0.072	<0.001	<0.001	0.004	<0.001	0.004
Maximum	0.68	0.118	0.239	2.354	0.007	0.003	0.089	0.002	0.104

Table 18.11: Times in seconds for the evaluation of the data set Karlsruhe City 1:2000.

	Conflict Graph	ILP1	ILP2	ILP3	GV	LIS	LID	FCS	FCD
Average	0.32	0.045	0.121	17.410	<0.001	<0.001	0.035	<0.001	0.025
Maximum	1.77	0.235	0.778	1645.0	0.025	0.026	0.751	0.021	0.414

Table 18.12: Times in seconds for the evaluation of the data set Karlsruhe City 1:4000.

	Conflict Graph	ILP1	ILP2	ILP3	GV	LIS	LID	FCS	FCD
Average	4.3	0.025	0.038	0.098	<0.001	<0.001	0.007	<0.001	0.006
Maximum	12.9	0.567	0.331	5.217	0.015	0.003	0.175	0.003	0.217

Table 18.13: Times in seconds for the evaluation of the data set Karlsruhe Countryside 1:2000.

	Conflict Graph	ILP1	ILP2	ILP3	GV	LIS	LID	FCS	FCD
Average	9.2	0.056	0.151	9.123	<0.001	<0.001	0.049	<0.001	0.035
Maximum	24.2	0.279	1.105	388.646	0.041	0.024	0.801	0.018	0.287

Table 18.14: Times in seconds for the evaluation of the data set Karlsruhe Countryside 1:4000.

19. Conclusion

In this part of the thesis we have shown that it is possible to find efficient ways to describe the trajectory based labeling problem (TBLP) as defined in Definition 14.2. To that end we introduced an appropriate model describing concepts as labels, conflicts between labels, visibilities of labels, viewports and trajectories of viewports.

Based on that model we defined for TBLP three different types of possible activity models (see Definition 14.4): We took up the Desiderata 13.1 introduced in [BDY06] and refined them in such a way that we also allow a label to become active and inactive when we can find a corresponding witness that makes the behavior of that label explainable for the observer of the map.

Further, we made the major decision to focus on polyarcs in order to describe trajectories, as those can be discretized easily by means of simple geometric tools. Based on those polyarcs we then explained how visibilities and conflicts of labels can be computed efficiently. For that purpose we extended the approach described in [GNR11]: The authors explain how conflicts between two rotating labels can be computed. We generalized their approach from labels to anchored rectangles such that the results also can be used for viewports. As byproduct we removed the restriction that the anchor of a label must coincide with a point of that label. Moreover, we successfully solved the problem to relate visibilities of labels to the corresponding trajectory such that we can compute intervals describing that visibility.

We also took care about the extensibility of the model. Thus, we introduced a weight function w in TBLP that helps to define new types of visibility without changing the model itself. For example in Section 15.4 we discussed how the notion of visibility of a label can be improved by also considering the conflict area between labels. We described in detail how by means of simple geometric tools the intersection area of two labels can be computed efficiently. Even the description was extensive, the result is limited to the redefinition of w such all other result that we have achieved are still applicable.

In Chapter 16 we also discussed theoretical aspects of TBLP: We proved that the problem is $\mathcal{NP}$-complete by a simple reduction from a static labeling problem and introduced a conflict graph based on the definition of TBLP that helps to formulate algorithms solving TBLP. In Chapter 17 we then presented based on that conflict graph ILPs for the different activity models and heuristics for the activity model AM1.

As we worked on an implementation of this part of the thesis, we also had the possibility to test the results on real world data. So far we only have considered simple heuristics (see Section 17.2) in order to cope with the $\mathcal{NP}$-completeness of TBLP. For sparse conflict graphs as they arise for low-scale maps (scale factor 1:2000) we could show that those heuristics are sufficient to solve TBLP with high quality (see Section 18.2). Surprisingly, we could ascertain that the choice of the applied heuristic influences the results only little. Thus, the heuristic GV lends itself to be the heuristic in favor, as its running time lies in $\mathcal{O}(n \cdot m \log(n \cdot m))$, where n is the number of labels and m is the number of arcs the trajectory consists of.

However, we could also show that for maps zoomed-out by a factor of 2 the solutions of the presented heuristics significantly differ from the optimal solution. On that account there is still space for research about more sophisticated heuristics that fill that gap.

Moreover, we have considered only heuristics so far, but we have not talked about ap-

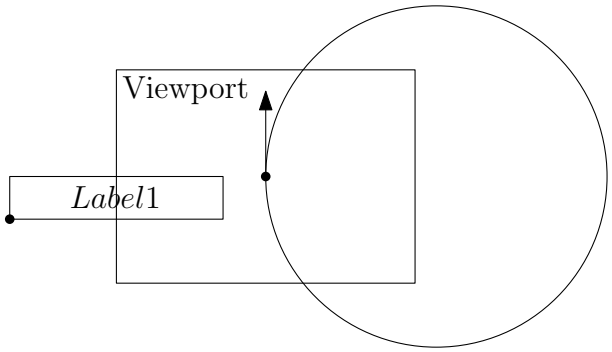


Figure 19.1: The size of the conflict graph depends on the size of the trajectory: Assume that the viewport moves m times along the circle, then the conflict graph contains m nodes for Label 1.

proximation algorithms with performance guarantee, yet: For similar labeling problems as presented in [GNR11] and [BNPW10] approximation algorithms are already known. For example in [GNR11] an efficient polynomial-time approximation scheme is presented for the case that one considers rotating maps. All of those approaches have in common that they extensively make use of the characteristic that the location of the labels induces an embedding for a corresponding conflict graph. Based on that embedding the problem is then divided into a pre-defined number of sub-problems that can be solved in constant time. Roughly spoken, those sub-problems consist only of constant many nodes because one can prove the number of nodes to be constant within a certain area.

It remains to answer the question whether those approaches can be adapted to find a similar approximation for TBLP. Up to now we can say that those approaches cannot be adapted straightforwardly: Although the labels induce an embedding for the conflict graph (see Definition 16.1) that is created based on labels and the considered viewport with trajectory, the problem arises that the number of nodes within a certain area also depends on the size of the trajectory: Each time the trajectory moves alongside a label at least one new visibility node is created in the corresponding conflict graph. In Figure 19.1 a corresponding example is illustrated: Assume that the trajectory consists of m cycles of the depicted circle, then the corresponding conflict graph contains m nodes for Label 1.

In the worst case it also could be that there is no good approximation algorithm for TBLP. By introducing ILP1 for the activity model AM1 we also shown that TBLP is closely related to maximum weighted independent set. In [Has96] it has been proven that there is no polynomial time algorithm that approximates maximum weighted independent set having a performance ratio of $n^{1-\epsilon}$ for any $\epsilon > 0$ unless $\mathcal{NP}$-hard problems have randomized polynomial algorithms. Consequently, one further question is whether for each general graph we can find a corresponding instance of TBLP. If that is the case the approximative hardness for maximum weighted independent set would also apply for TBLP.

Summarizing, we have introduced an appropriate model with different variants for the trajectory based labeling problem such that we can obtain the visibility of labels and their conflicts efficiently. Further, we could present both theoretical and practical results that help to cope with TBLP. However, there is still research to do for describing and analyzing TBLP completely.

Bibliography

[AGT86] D. Avis, T. Gum, and G. Toussaint. Visibility between two edges of a simple polygon. *The Visual Computer*, 2:342–357, 1986. 10.1007/BF01952419.

[AS93] P.K. Agarwal and M. Sharir. Circle shooting in a simple polygon. *Journal of Algorithms*, 14(1):69 – 87, 1993.

[BDY06] K. Been, E. Daiches, and C.E. Yap. Dynamic map labeling. *IEEE Trans. Vis. Comput. Graph.*, 12(5):773–780, 2006.

[BNPW10] K. Been, M. Nöllenburg, S.H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry*, 43(3):312 – 328, 2010. Special Issue on 24th Annual Symposium on Computational Geometry (SoCG'08).

[Bol75] K. Bolton. Biarc curves. *Computer Aided Design*, 7(2):89–92, 1975.

[CC09] P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 892–901, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[CG89] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. *Discrete Comput. Geom.*, 4(6):551–581, September 1989.

[CW95] S.Y. Chou and T.C. Woo. A linear-time algorithm for constructing a circular visibility diagram. *Algorithmica*, 14:203–228, 1995.

[dBvKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[DRS08] R.L.S. Drysdale, G. Rote, and A. Sturm. Approximation of an open polygonal curve with a minimum number of circular arcs and biarcs. *Computational Geometry*, 41(1–2):31 – 47, 2008. Special Issue on the 22nd European Workshop on Computational Geometry (EuroCG).

[EW91] P. Egyed and R. Wenger. Ordered stabbing of pairwise disjoint convex sets in linear time. *Discrete Applied Mathematics*, 31(2):133–140, 1991.

[FL91] Th. Fraichard and C. Laugier. Smooth trajectory planning for a car-like vehicle in a structured world, 1991.

[FW91] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the seventh annual symposium on Computational geometry*, SCG '91, pages 281–288, New York, NY, USA, 1991. ACM.

[GHMS91] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, and J.S. Snoeyink. Approximating polygons and subdivisions with minimum-link paths. *Internat. J. Comput. Geom. Appl*, 3:383–415, 1991.

[GNR11] A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, volume 6844 of *Lecture Notes in Computer Science*, pages 451–462. Springer Berlin / Heidelberg, 2011.

[Has96] J. Hastad. Clique is hard to approximate within ... In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, FOCS '96, pages 627–, Washington, DC, USA, 1996. IEEE Computer Society.

[HE05] M. Held and J. Eibl. Biarc approximation of polygons within asymmetric tolerance bands. *Computer-Aided Design*, 37(4):357–371, 2005.

[KM03] G.W. Klau and P. Mutzel. Optimal labelling of point features in rectangular labelling models. *Mathematical Programming*, 94:435–458, 2003.

[KOHH05] A. Kako, T. Ono, T. Hirata, and M.M. Halldórsson. Approximation algorithms for the weighted independent set problem. In *IN GRAPH-THEORETIC CONCEPTS IN COMPUTER SCIENCE, 31ST INTERNATIONAL WORKSHOP, WG*, pages 341–350, 2005.

[KS95] D. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. In Selim Akl, Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 183–193. Springer Berlin / Heidelberg, 1995.

[LSC08] M. Luboschik, H. Schumann, and H. Cords. Particle-based labeling: Fast point-feature labeling without obscuring other visual features. *IEEE Transactions on Visualization and Computer Graphics (InfoVis'08*, pages 1237–1244, 2008.

[MS91] J. Marks and S. Shieber. The computational complexity of cartographic label placement. Technical report, 1991.

[OvL81] M.H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, pages 166–204, 1981.

[PBP02] H. Prautzsch, W. Boehm, and M. Paluszny. *Bezier and B-Spline Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[PGP03] I. Petzold, G. Gröger, and L. Plümer. Fast screen map labeling – data structures and algorithms. In *IN 21ST INT. CARTOGRAPHIC CONF*, pages 288–298, 2003.

[PPH99] I. Petzold, L. Plümer, and M. Heber. Label placement for dynamically generated screen maps. In *Proceedings of the Ottawa ICA*, pages 893–903, 1999.

[Pre79] F. P. Preparata. An optimal real-time algorithm for planar convex hulls. *Commun. ACM*, 22(7):402–405, July 1979.

[SH76] M.I. Shamos and D. Hoey. Geometric intersection problems. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, SFCS '76, pages 208–215, Washington, DC, USA, 1976. IEEE Computer Society.

[vKSW99] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13:21–47, 1999.