Karlsruhe Institute of Technology

Three Dimensional Dynamic Point Labelling

Master Thesis of

Erik Prause

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Prof. Dr. rer. nat. Dorothea Wagner
	Prof. Dr. rer. nat. Peter Sanders
Advisors:	Dipl.-Inform. Andreas Gemsa
	Dipl.-Inform. Benjamin Niedermann
	Dr. Martin Nöllenburg

Time Period: 29th November 2013 – 28th May 2014

Acknowledgements

First of all, I want to thank my supervisors Andreas Gemsa, Benjamin Niedermann and Martin Nöllenburg for their great support during the past few months. Their guidance, experience and advices were essential for my work to come about. They welcomed me, when I first came up with this ambitious topic and showed patience and reassured me during some long periods of unfruitful research.

I want to thank Prof. Dr. Dorothea Wagner for giving me the possibility to write my master thesis at her Institute, for hospitality and grading this work.

I would like to give special thanks to Sandra, who supported me mentally and without whom it would have been way harder to write this thesis. I would also like to thank especially my parents and my sister for believing in me. It is their support that brought me that far.

My four fellow students of the learning group deserve special consideration, who accompanied me from my first day at the Universität Karlsruhe (TH) years ago up to the end of my studies at the KIT. Only together with them I was able to gather all the experience and knowledge necessary for this work.

Finally I would like to thank my substitute family M6 for a warm welcome home every evening and providing distraction to get my thoughts off of work whenever necessary.

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 28th May 2014

Deutsche Zusammenfassung

Die Darstellung dreidimensionaler Information hat, nicht zuletzt dank moderner Visualisierungstechniken, viele Anwendungsgebiete erobert. Im Bereich der Visualisierung von 3D-Datenpunkten trägt neben der topologischen Struktur oftmals zusätzliche textuelle oder bildliche Information zum Verständnis eines dargebotenen Sachverhaltes bei. Die Information wird mittels eines Labels mit dem zugehörigen Datenpunkt in Verbindung gebracht, welches ebenfalls auf der Bildebene angezeigt wird.

Die Möglichkeit, eine 3D-Darstellung dynamisch erkunden zu können, erhöht das Verständnis der dargestellten Information, bringt aber neue technische Herausforderungen mit sich. Von einer dynamischen Transformation der Darstellung, wie zum Beispiel dem Zoomen, hat der Nutzer inherent ein kontinuierliches Verständnis. Entsprechend ist es erforderlich diese Kontinuität bei der Label-Platzierung zu berücksichtigen.

Um eine hohe Qualität der Lösung zu garantieren, ist es notwendig die dynamischen Eigenschaften der Label-Platzierung mittels mathematischer Modelle zu untersuchen und somit die Qualität einer Lösung erst formulieren zu können.

Diese Arbeit stellt einen ersten theoretischen Ansatz vor, das Labeling dreidimensionaler Datenpunkte mittels Zoomen zu dynamisieren. Basierend auf der projektiven Darstellung von dreidimensionalen Datenpunkten auf einer zweidimensionalen Bildebene wird eine Methode zur Platzierung von Labels in einem diskreten Ein-Positions-Modell präsentiert. Die Dynamik des Zoomens wird durch die Bewegung der virtuellen Kamera entlang einer festen Koordinatenachse modelliert und anhand dessen die Güte einer Lösung mathematisch charakterisiert. Aufbauend darauf werden zwei verschiedene Platzierungsmodelle für Labels vorgestellt, für welche NP-Schwere bewiesen wird. Während für das erste Modell sowohl gemischt-ganzzahlige Optimierungsprobleme, als auch heuristische Lösungsangsätze präsentiert werden, wird für das zweite Modell schrittweise ein Approximationsalgorithmus mit konstantem Approximationsfaktor und polynomieller Laufzeit aufgestellt.

Contents

List of Figures

1. Introduction

One of the most deeply rooted urges of mankind is his curiosity. He always aspired towards gathering knowledge and understanding the world around him. This striving culminates in modern science with its vast number of different, very specialized disciplines and research fields.

Research nowadays takes place on subjects of very different scales and natures, from (sub-) molecular phenomena to astronomic objects and constellations. From mastering practical applications as for example medical imaging or designing engines, to quantifying rather elusive subjects such as the human psyche.

Scientific work at such a high level generates a need for sophisticated tools to ease analysis or even permit it in the first place. An important task of computer science is to provide and improve such tools to expose the information inherent to statistical data or visualize the structure of spatial phenomena.

There are very different aspects to visualisation of scientific data, which for example appear in terms of 2D or 3D data points. On one hand, it is not enough to just plot data points statically and leave it at that. But rather the necessity (and with modern computer technology also the means) emerges to evolve dynamic display tools that allow researchers to inspect data from different points of view, zooming, rotating and moving visualisations at their needs.

On the other hand, for thoroughly understanding the information provided by the data points, it is necessary to not just see their topological relation, but also individually identify them or their properties. Therefore the idea comes to mind to place labels besides data points that provide this additional information.

The utilization of such visualization tools is not restricted to scientific purposes only. A vast variety of possible applications opens up, as there is for example need for dynamic labelling in Computer Aided Design (CAD), in statistical analysis tools for economics or for automated annotation of medical imaging as well as several use cases in consumer electronics.

For labelling 2D data points, much research work has been done recently, mainly in the scope of labelling locations on maps. There exist theoretical as well as practical works on static as well as dynamic labelling.

On dynamic labelling of 3D data points, however, there seems to be done less research up to now. There exist several practical approaches, which are rather heuristic and lack guarantees on optimality. With this work, we want to take a first step on a theoretical approach for this topic.

In particular we investigate how labels can be attached to data points in 3D that are presented on a dynamically zoomable viewport. We examine the data points' and labels' geometric properties with the aim of placing them in a non-overlapping way. This leads to a problem formulation whose solution we characterize in terms of optimal usability and in terms of theoretical complexity. We present different approaches to actually finding optimal solutions and giving approximation guarantees for sub-optimal solutions.

We choose a theoretically consolidated approach towards dynamic labelling of 3D data points. Since this is a first approach towards this topic, we concentrate on zooming only to be able to refine a sound, clear problem formulation. This simplified model, however, allows us to obtain fundamental theoretic results. We provide two different labelling models and according optimization problems and show NP-hardness for both of them. For the first of the two, we state MILP (mixed integer linear programming) and ILP (integer linear programming) formulations, as well as heuristics. Based on findings gained from that, we provide a novel method to solve the second of the two models. We introduce the Radial Stabbing Tree and use it to derive a constant factor approximation algorithm for the dynamic labelling problem for 3D data points.

1.1 Related Work

In the context of algorithmic map generation, static labelling as been investigated thoroughly in past research. Especially for point feature labelling highly sophisticated models have been developed, the most basic of which is the fixed position labelling model, in which the label may only be placed at a finite number of discrete positions relative to the corresponding data point. In the slider model proposed by van Kreveld et al. [vKSW99], the label may be placed in a continuous range of positions relative to the data point.

The global placement can be influenced in the selection of a subset of labels to be displayed and in the position of each label relative to its data point. It may optimized in the number of labels displayed simultaneously without overlapping each other, which is known as MaxNum labelling problem. Even relatively simple versions of this problem, as well as the MaxSize labelling problem mentioned in the following, are proven to be NP-hard.

Formann and Wagner introduced a MaxSize labelling problem [FW91], in which all labels are displayed without overlapping by scaling the size of the labels by a common stretching factor so that they do not intersect with surrounding labels. They ask for the maximum stretching factor, for which all labels can be placed without intersection and provide an approximation algorithm to answer this question.

Also static labelling approaches have been investigated, in which all labels are placed at their original size and the number of intersections is minimized only by changing the positions of the labels relative to their corresponding data points, as for example by Gerrits et al. [dBG12]. This problem class is referred to as MaxFree labelling problem.

Dynamic labelling, opposed to static, is concerned with placing labels on a display, onto which continuous operations such as panning, zooming or rotation are applied. The groundwork on dynamic labelling has been done by Been et al. in [BDY06] and [BNPW10] at the scope of 2D map labelling. They introduce general labelling models as well as criteria for a good label layout. Those criteria are defined in terms of the consistency of a dynamic map labelling by introducing restrictions to when a label is allowed to appear and

disappear during continuous display transformations. They aim to avoid distraction of the user due to flickering and unexpected or improper label placement.

Been et al. concentrated their work on panning and zooming of maps only. This work was extended by Gemsa et al. [GNR11] who investigate on rotation of the map while keeping the labels upright and by Niedermann et al. [GNN13] to allow for free display movement along continuous trajectories.

Dynamic 3D labelling has been studied in the context of different applications as for example medical imaging [RPRH07, MP09, AHS05] and labelling of buildings in virtual city models [MD06a, MD06b, MD08] to name only two.

Herein, different labelling layouts have been developed, like attaching the labels directly to their corresponding object [MD06a, MD08], connecting the labels via line segments, so called leaders [uB10, HAS04] or hybrid methods aiming for the benefits of both approaches [RPRH07, GHS06].

The type of labelled objects ranges from single data points [SD08], to areas [uB10, HAS04] and up to complete 3D objects to be labelled [RPRH07]. The labels itself can be placed in image space [HAS04] or in world space [RPRH07, MD06a], exploiting the spatial structure of the 3D objects.

Very different aspects of dynamic labelling are investigated; some solutions allow only one degree of freedom for the camera as for example zooming [PRS97] whereas other works are published on embedding information into a VR-environment (virtual reality) [AF03, BFH01], which allows the camera to be moved freely in 3D space. This high degree of freedom makes reactive label placement necessary and does not allow for precalculated solutions. Goetzel et al. investigate in [GHS07] on labelling dynamic 3D scenes with animated objects, which they solve by searching for calm areas in the animation into which they place static anchor points for leader-connected labels.

However, all of the above approaches are heuristic or experimental. Some formulate a metric on optimality, but none gives an optimal solution or even an approximation guarantee for the solution. Dynamic 3D labelling is a very important topic, yet little to no fundamental theoretic work seems to have been done on this field of research.

A rather formalized approach has been introduced by Gerrits et al. [dBG13]. Although the algorithm is intended for labelling dynamic 2D maps where data points move arbitrarily, this method can be utilized to label the dynamic projection of 3D data points on a screen. But this approach is still heuristic and, as it realizes the MaxFree labelling problem, does not guarantee intersection free placement of labels.

1.2 Outline

In this thesis, we want to propose methods for dynamically labelling 3D data points. Our models allow the user to zoom in and out on a bounded zooming ratio interval while the maximum possible number of labels is displayed.

For this purpose we introduce in Chapter 2 the geometric and mathematical tools to dynamically display labels and analyse intersections between them. This provides the groundwork for the formulation of the Active Range Optimization problem at the end of this chapter.

In Chapter 3 we derive a first labelling model and prove its NP-hardness. To further investigate on this model, we develop MILP and ILP formulations, as well as heuristics, which are then evaluated in experiments.

On this basis, a second labelling model is developed in Chapter 4, for which we obtain an optimal, polynomial time algorithm on a 1D screen. However this problem is also NP-hard on a 2D screen. For this second case, we introduce a novel Radial Stabbing Tree to eventually be able to propose a polynomial time, constant factor approximation algorithm.

In Chapter 5 we draw a conclusion and give further prospects for successive research.

2. Dynamic Display Model

In this chapter, we will introduce the basic mathematical methods for displaying 3D data points, providing them with labels and characterizing intersections between labels, both, for static and dynamic labelling models. For this purpose we will at first have a glance at projective transformations, i.e., how to display 3D points on a 2D screen. Then we will continue to describe how labels can be attached to those points. At this point, we will also investigate how intersections between labels can be characterized. After that, we will have a look at zooming transformations and how they influence the projection of points and intersection of labels. Finally, we will reduce this geometric characterization of label intersections to a combinatorial label placement problem.

2.1 Projection of Points in 3D: The Pinhole Camera

Initially, we are given a set $S \subset \mathbb{R}^3$ of data points in 3D-space which are to be presented to the user. That could be for example a 3D medical data set obtained through medical imaging or a map of astronomical objects in the galaxy. Throughout this work we will assume general position of the data points to avoid odd case-by-case analysis and improve the comprehensibility of this thesis. Still all results and algorithms can easily be adapted to cope with in any way degenerated inputs.

To present those 3D points in S on a 2D screen, they somehow have to be projected onto a plane, the so called *viewplane*. A very common type of projection in computer graphics is the *perspective projection* [SM09]. It allows for a good understanding of the structure of the scene in 3D space and coincides relatively well with the human perception.

The perspective projection is achieved by means of a (theoretic) *pinhole camera* model. The pinhole camera is thought of as a small box into which light can only pass through a tiny hole (the pinhole) in one side of the box. The hole is so small, that from each direction only one ray of light can pass through (cf. Figure 2.1). The rays of light passing through the pinhole will result in an flipped projection of the box's surroundings inside the box on the wall opposed to the hole. This perfect pinhole camera model does not exactly depict real world cameras or the human perception, but still approximates it well enough.

Mathematically, the projection is described by the projective space $\mathbb{P}^2(\mathbb{R})$ which is defined by a set of equivalence classes in $\mathbb{R}^3$. All points of $\mathbb{R}^3$ that lie on a common line through the origin, except for the origin itself, are regarded equivalent. Such an equivalence class is represented by any of it's members, especially by the point at which this line intersects with

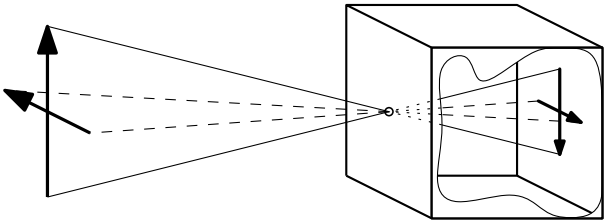


Figure 2.1: The basic setup of a theoretic pinhole camera.

the view plane. This definition agrees with the intuitive idea that all points on the same line of sight are projected to the same point on the screen and after being projected, we cannot determine its distance from the screen without further information. This mathematical model assumes the center of projection of the pinhole camera to be located at the origin of $\mathbb{R}^3$.

The reason for us using the pinhole camera model is its widespread usage in computer graphics and its simple geometric representation. We can use the intercept theorem to easily calculate the projection of a point in $\mathbb{R}^3$ onto the image plane.

We assume the pinhole camera (or rather the pinhole itself and therefore the center of projection, cf. Figure 2.2) to be located at $(0, 0, z_c)$ and directed in negative z-direction. If it wasn't we could easily transform the whole scene including the camera by first translating it by $(-x_c, -y_c, 0)$ with (x_c, y_c, z_c) being the actual camera location, and then rotating the scene around the point $(0, 0, z_c)$ so that the view direction of the camera is aligned with $(0, 0, -1)$.

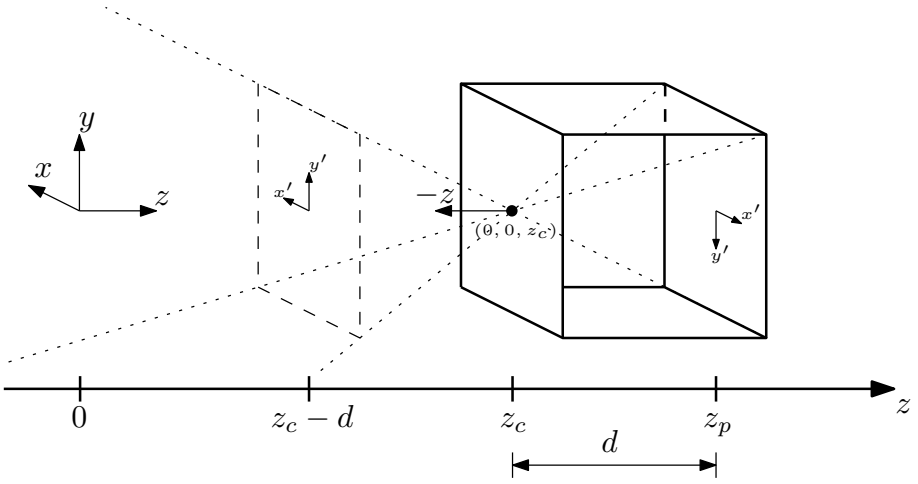


Figure 2.2: Camera coordinates and direction in the pinhole camera model.

The plane onto which the scene in front of the camera is projected, is located behind the camera with its center at $(0, 0, z_p)$ with distance $d = z_p - z_c > 0$. The projection onto this view plane will be upside-down and mirrored horizontally as described earlier. Instead of using this 'real' flipped view plane, we will use a 'virtual' viewplane in front of the camera at $(0, 0, z_c - d)$ (cf. Figure 2.2), which has the same distance from the center of projection and the same extent as the original one, but has the advantage of not being flipped.

Be $a = (x_a, y_a, z_a) \in \mathbb{R}^3$ a data point which is to be projected. Since only points in front of the camera can be projected and the camera is looking in negative z-direction, we will assume that $z_a < z_c$ and all of the following equations only apply if $z_a < z_c$.

To calculate the projection of the point a we make use of the intercept theorem (cp. Figure 2.3) by

$$\frac{x'_a}{x_a} = \frac{z_p - z_c}{z_c - z_a} \tag{2.1}$$

$$x'_a = x_a \cdot \frac{z_p - z_c}{z_c - z_a} \tag{2.2}$$

$$= x_a \cdot d \cdot \frac{1}{z_c - z_a}, \tag{2.3}$$

$$y'_a = y_a \cdot d \cdot \frac{1}{z_c - z_a}, \tag{2.4}$$

$$z'_a = z_c - d \tag{2.5}$$

$$\text{for } z_a < z_c \tag{2.6}$$

which yields the projected point $a' = (x'_a, y'_a) \in \mathbb{R}^2$.

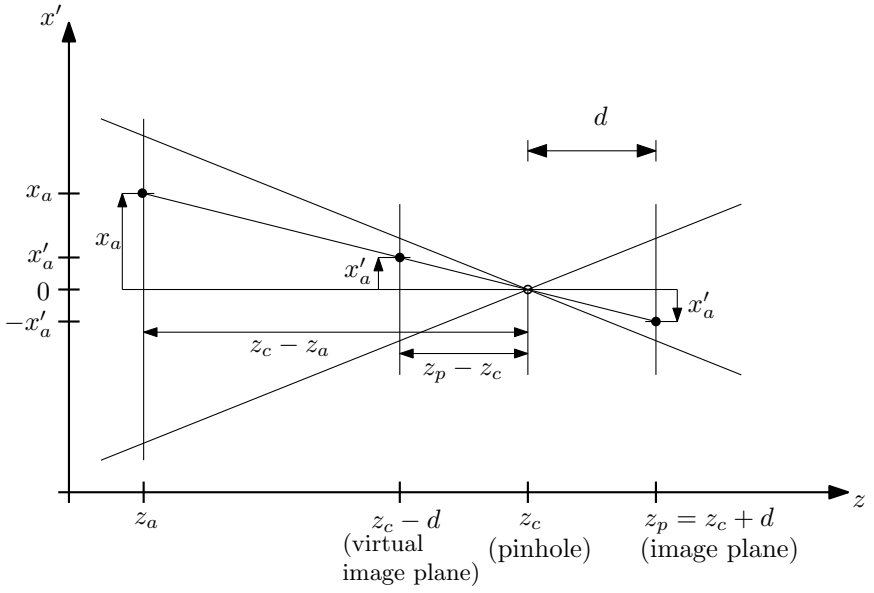


Figure 2.3: The intercept theorem in the pinhole camera model: $\frac{x'_a}{x_a} = \frac{z_p - z_c}{z_c - z_a}$. Figure shows x'-coordinate only, y'-coordinate equivalent.

2.2 Attaching Labels

With each data point a label is associated, which could be for example a text or a pictogram that identifies the point or gives additional information. In addition to the data points, the labels must also be displayed on the screen to make their information accessible to the user. We represent each label by its rectangular, axis aligned bounding box, which is describable by a width and a height. This extent is measured in screen coordinates, i.e. its apparent size on the screen.

There exist several different labelling layouts, i.e. methods to attach the label to its corresponding data point. For example, we could render the labels anywhere on the screen and connect them to their data points using a straight line [AF03]. Another method, used in [BHKN08], is to place all labels at the screen's border and connect them to their data points via paths of horizontal and vertical line segments.

In this work we will focus on rendering the label directly besides the corresponding data point, so that the data point is covered by the label or its outline. Here again different possible layout options exist.

In literature, researchers most commonly distinguish between laying at discrete positions or at continuous ranges of positions. In [Ger13] for example, the authors use a 4-slider model with rectangular labels, which allows the label to be placed at any position, as long as the corresponding projected data point is located at the label's border. This model is named 4-slider model because the rectangular label can be 'slid' along four continuous intervals of positions and is therefore a continuous positioning model.

Discrete position models, on the other hand, only consider a limited set of allowed positions, relative to the data point, at which the label can be placed. The most restrictive, yet most simple set of positions would result from allowing only one label position; this is the so called 1P-model [dBG12]. We will in the following take a closer look at this simple model. More precisely, we allow the label to be placed right of and above the data point only, so that the data point (in relation to the label) is located at the lower left corner of the label. This location of the data point relative to the label is called the label's anchor point (cf. Figure 2.4). The location of the anchor point therefore is equal to the projection of the data point.

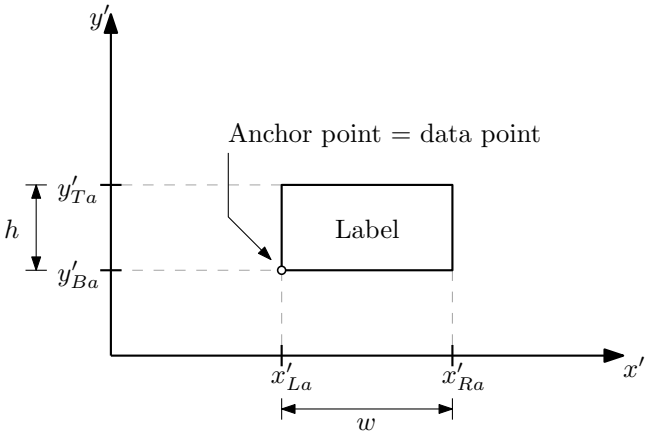


Figure 2.4: A labelled data point, the label's anchor point and its borders.

The label is supposed to keep its on-screen size and orientation, despite of any operations manipulating the screen like zooming, rotation or translation. By postulating this constraint, we want to achieve that the label is well readably presented to the user, no matter the screen's state.

So if we want to draw the rectangular labels together with their data points on the screen, we first need to find a mathematical description of the label's left, right, bottom and top borders with respect to its associated data point. Since the label is supposed to keep its given size and orientation despite any screen operations, it will not be projected using the above interception formula. Instead, we will calculate its on-screen location relative to the data points projection, which is equal to the label's anchor point, depending on the label's extent.

Given a data point a and its projection $a' = (x'_a, y'_a)$ the label itself is drawn at a' with width w and height h. Since the label should be drawn right of and above the data point, a' is equal to the bottom left corner of the label (i.e. its anchor point):

$$x'_{La} = x'_a = x_a \cdot d \cdot \frac{1}{z_c - z_a} \tag{2.7}$$

$$y'_{Ba} = y'_a = y_a \cdot d \cdot \frac{1}{z_c - z_a} \ . \tag{2.8}$$

As the extent of the label is given in screen coordinates, there is no need for any transformations and therefore the right and the top borders of the label at offsets w and h are given by

$$x'_{Ra} = x'_a + w = x_a \cdot d \cdot \frac{1}{z_c - z_a} + w \tag{2.9}$$

$$y'_{Ta} = y'_a + h = y_a \cdot d \cdot \frac{1}{z_c - z_a} + h \ , \tag{2.10}$$

see Figure 2.4.

2.3 Zooming and Dynamic Projection

Static labelling shows an invariant data set from a fixed perspective, i.e. only a certain state of the whole scene and the user's view of it. Therefore 3D static labelling is basically the same as 2D static labelling, except for the projection of the data points. That is, if we were to present 3D data points statically and label them (statically), we would first project all the data points onto the view plane and then apply a 2D labelling algorithm to render the labels, for example by use of the algorithm presented in [vKSW99].

If we wanted the user to be able to interact with the scene or its representation, we would have to add certain input dynamics to the model. Dynamic labelling, opposed to static labelling, is concerned with finding a somehow optimal labelling while the user is allowed to manipulate the scene or its projection.

This user interaction is supposed to be some kind of continuous function applied to the viewport. For example when allowing the user to rotate the viewport, this could be realized by a parameterized function rotating the data points around the center while keeping the labels upright and anchored at their corresponding data point. The input parameter of this function could be the current rotation angle, so that user interaction itself now is reduced to choosing the right parameter (rotation angle) and then evaluating the precalculated labelling function for this parameter.

In most cases of the previous work only one kind of user interaction is assumed and the optimal label placement (-function) is precalculated for the whole allowed input parameter range for this user interaction.

For example in [GNR11] the authors allow the user to rotate the screen while as many labels as possible are to be displayed without overlapping. First the solution is calculated for all rotation angles. The user then can freely rotate the screen; his interaction is translated into a one dimensional parameter encoding the current rotation angle. The labels are placed by evaluating the precalculated labelling function with this parameter.

Another instance of dynamic label placement is discussed in [BNPW10]. The authors present a dynamic labelling method for a two-dimensional map regarding zooming operations applied to it. They aim for labelling functions that place as many labels as possible for a continuous range of the zooming ratio while no two labels overlap. This is, same as in the previous example, achieved by precalculating an overall solution for the complete available zooming interval. The map then is presented to the user allowing continuous zooming operations by evaluating the precalculated labelling function for the current zooming ratio.

We will now take a closer look on how zooming operations can be expressed as a parameterized function for 3D data points and how those operations influence the placement of labels.

Zooming itself will be modelled by moving the camera along the z-axis. Since the camera is looking in negative z-direction, zooming *in* is equivalent to decreasing the cameras

z-position z_c (i.e. moving into the current field of view) and zooming *out* is equivalent to increasing it (i.e. moving away from the current field of view).

That means the user's input commands for zooming are translated into a one dimensional parameter representing the camera's current z-position. When the input changes, the z-coordinate z_c of the center of projection is altered. Taking equations 2.2 ff. and 2.7 ff., the projection of a point a and the borders of its label can be described as functions of z_c:

$$x'_a(z_c) = x_a \cdot d \cdot \frac{1}{z_c - z_a} \tag{2.11}$$

$$y'_a(z_c) = y_a \cdot d \cdot \frac{1}{z_c - z_a} \tag{2.12}$$

$$x'_{La}(z_c) = x'_a(z_c) = x_a \cdot d \cdot \frac{1}{z_c - z_a} \tag{2.13}$$

$$y'_{Ba}(z_c) = y'_a(z_c) = y_a \cdot d \cdot \frac{1}{z_c - z_a} \tag{2.14}$$

$$x'_{Ra}(z_c) = x'_a(z_c) + w = x_a \cdot d \cdot \frac{1}{z_c - z_a} + w \tag{2.15}$$

$$y'_{Ta}(z_c) = y'_a(z_c) + h = y_a \cdot d \cdot \frac{1}{z_c - z_a} + h \tag{2.16}$$

All the trajectory functions in equation 2.11 to 2.16 are of the form

$$f(z) = \alpha_f \cdot \frac{1}{\beta_f - z} + \gamma_f \tag{2.17}$$

i.e. simple reciprocal functions with parameters α_f, β_f and γ_f. The trajectory of the projected point on the screen is a straight line through the origin as depicted in Figure 2.5. The corners of the rectangular label move on lines parallel to the point's projection trajectory.

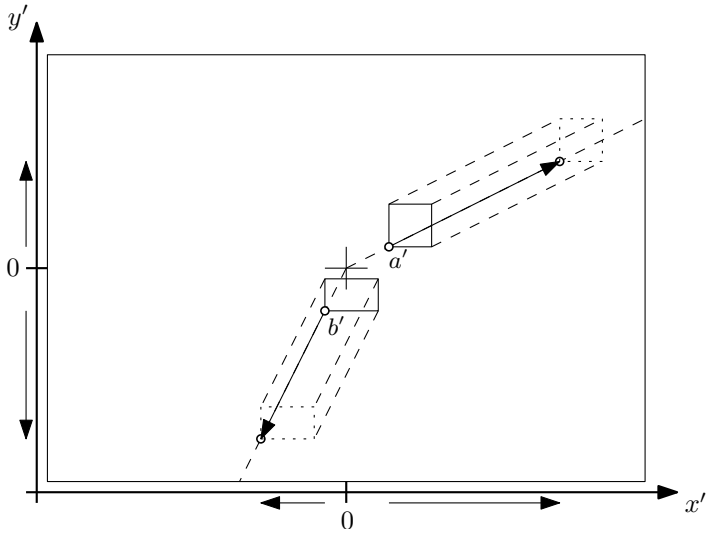
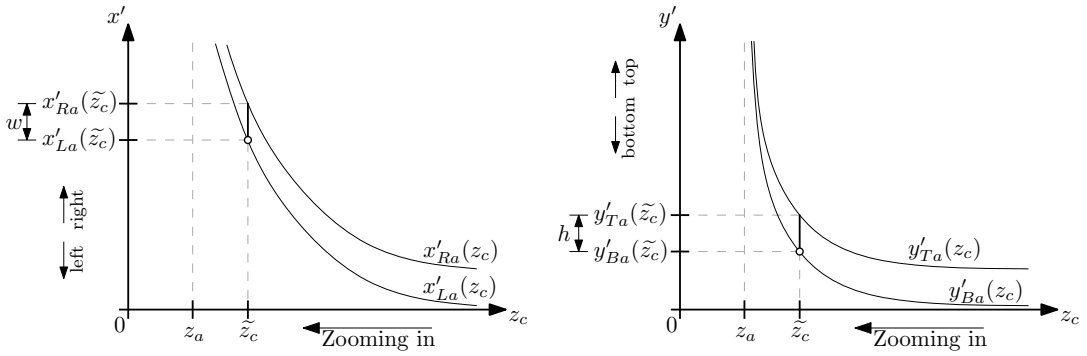


Figure 2.5: The on-screen trajectories of two data points and their labels. Arrows depict movement direction of labels when zooming in.

For the data points' trajectories (equations 2.11 f.), the offset parameter is $\gamma_f = 0$, i.e. the projection function has no vertical or horizontal offset. Hence, for $z_c \to \infty$, which is equivalent to zooming out very far, the projections of all data points approach $(x', y') = (0, 0)$, the center of the projection plane (cf. Figure 2.5 and Figure 2.6). The left and bottom borders of the label (equations 2.13 f.) directly follow their data point's

trajectory, since the label is directly attached to the data point with its left lower corner. I.e. the corresponding projection functions have exactly the same form as the data point. In contrast to this, the right/upper label border has an offset from the data point that equals the width/height of the label, i.e. the offset parameter is $\gamma_f = w$ and $\gamma_f = h$ respectively. The remaining parameters α_f and β_f are equal to those of their corresponding lower bound (see Figure 2.6) which means the functions have exactly the same shape, only with a different (vertical) offset.



(a) x'-projection trajectory of label a

(b) y'-projection trajectory of label a

Figure 2.6: The label's projected x'- and y'-position relative to the current zooming ratio z_c.

We stated in equation 2.6 that the projection of points only is defined for $z_c > z_a$. The functions' parameter β_f is equal to z_a and this is exactly where $f(z)$ has a singularity and is not defined, cp. Figure 2.7. Accordingly, $f(z)$ is only defined for $z > \beta_f$; the complete part of the function for $z \leq \beta_f$ ("left" to the singularity) is not valid for placing the label and hence irrelevant. We will regard the function for $z > \beta_f$ only, that means for projecting the data point and its label dynamically for a given zoom state z, we only display labels (and therefore only evaluate $f(z)$), if $z > \beta_f$.

Parameter α_f finally influences the slope of the curve. Due to this parameter, labels will move with different speed on the screen. Furthermore, if α_f is negative, the trajectory will be bent "downwards". That means the label moves into negative x' direction, but still away from the center of the screen.

We can now use this formulation for a first simple dynamic label placement. Zooming operations are translated into moving the camera forth and back along the z-axis and therefore yielding an input parameter z_c. We then evaluate the above dynamic projection functions for all data points and their labels and render them all together to the screen.

For dense data points, this would certainly result in labels overlapping each other and the information presented by those labels becomes inperceivable for the user. Instead we have to resolve conflicts between the labels, which we achieve by choosing a non-overlapping subset of labels to be rendered, so that the user can at least read the information given by this subset. The problem of finding a subset that is as big as possible but still satisfies given constraints is known as *MaxNum* optimization problem.

Before finding a solution on which labels to choose to be rendered, we will have to investigate on how to characterize intersections between labels.

2.4 Dynamic Label Intersections

Intersections between two labels occur, when two labels occupy the same space of the screen at the same time. Hence, considering the dynamic projection and labelling model

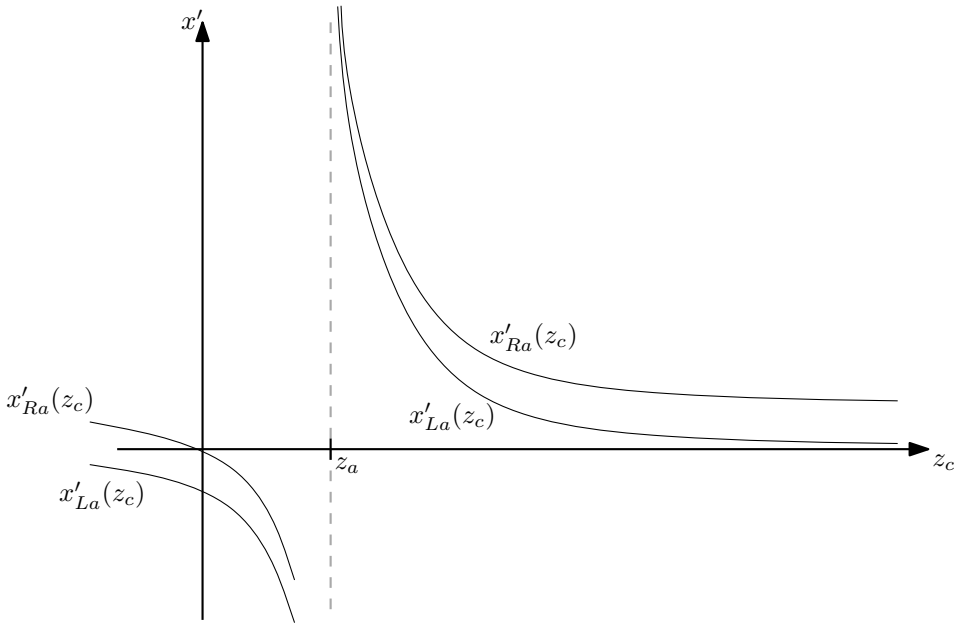


Figure 2.7: A label's zooming trajectory. The part of the function left to the singularity is not valid for label placement.

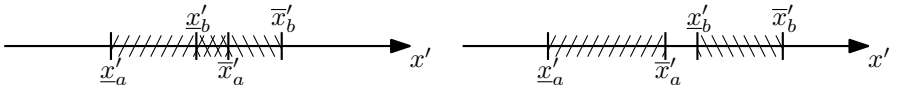
presented above, we are interested in the zooming ratio ranges for which two labels occupy the same space of the screen.

To approach this question, we will at first reduce the projective plane to only one dimension. The screen therefore becomes a line instead of a plane and the labels are intervals on that line. This means we will only regard one of the trajectory diagrams in Figure 2.6 and intersections between the trajectories depicted in it; more precisely, we will concentrate on the x'-projection. Since the screen is no more a 2D plane, we call this linear screen the *1D screen* and the projection to it *1D projection*.

Now, overlapping of two labels is equivalent to two labels' projection intervals intersecting. Given two label projection intervals $x'_a = [\underline{x}'_a, \overline{x}'_a]$ and $x'_b = [\underline{x}'_b, \overline{x}'_b]$ of labels a and b, the labels intersect only if

$$\underline{x}'_a \leq \overline{x}'_b \wedge \overline{x}'_a \geq \underline{x}'_b \tag{2.18}$$

cf. Figure 2.8.



(a) Projection intervals do intersect. (b) Projection intervals do not intersect.

Figure 2.8: Intervals of label projections intersect on 1D screen.

Using the notation of the labels' left and right borders (see equations 2.7 and 2.9) we can rewrite this statement to

$$x'_{La} \leq x'_{Rb} \wedge x'_{Ra} \geq x'_{Lb} \tag{2.19}$$

for two labels at data points a and b.

As we stated earlier, the projection of data points move with differing speed on the screen when zooming in or out. This difference in movement speed implies that the relative

differences between the points' positions can change. Therefore, the question whether there is an intersection between two labels, depends on the current zooming ratio. We will therefore investigate for which zooming ratios, the labels intersect, i.e. for two labels a and b we are interested in the set

$$\{z_c \in \mathbb{R} \mid x'_{La}(z_c) \leq x'_{Rb}(z_c) \wedge x'_{Ra}(z_c) \geq x'_{Lb}(z_c)\} \ . \tag{2.20}$$

The labels' zooming trajectories intersect like depicted in Figure 2.9, so that if we want to know for which zooming ratios the labels overlap, we have to calculate the intersection between those trajectories.

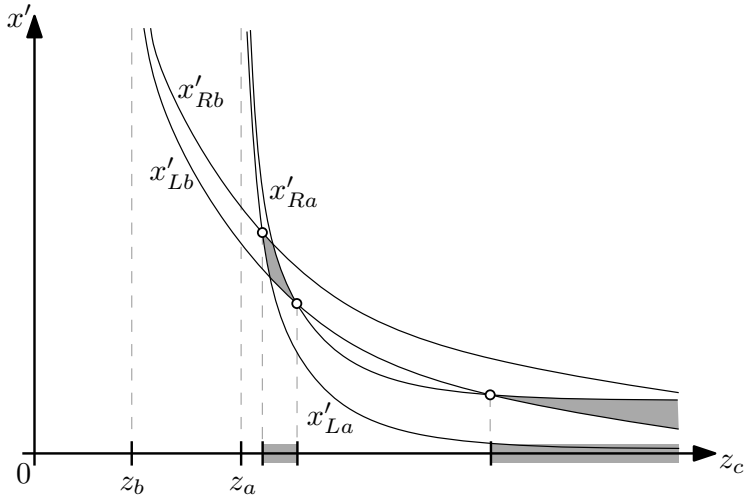
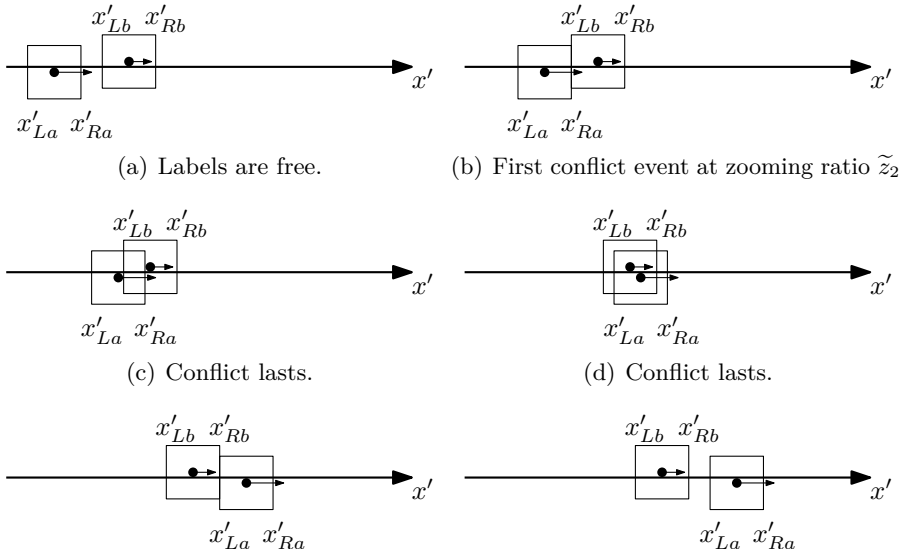


Figure 2.9: Trajectories of two labels, which intersect for different zooming ratios. Conflicts are marked grey.

When two labels at projected points a' and b' overlap for a certain range of the zooming ratio, the intersection starts when overlapping occurs between the left border of one, and the right border of the other label, i.e. at $x'_{La}(z_c) = x'_{Rb}(z_c)$ and $x'_{Lb}(z_c) = x'_{Ra}(z_c)$ as depicted in Figure 2.10. We call the intersection between the labels' borders a *conflict event*. Only at those conflict events the two labels can enter a mutual conflict or leave it, because the label movement on the screen is continuous. The zooming ratio range in between two conflict events, for which the labels actually have a conflict, is called *conflict interval*.

Figure 2.11 depicts the labels' trajectories while moving the center of projection z_c along the z-axis, which is equivalent to zooming. In this example we can see more clearly, that conflict intervals start/end when the trajectory of the left border of one and the trajectory of the right border of the other label intersect.



(a) Labels are free.

(b) First conflict event at zooming ratio $\widetilde{z}_2$.

(c) Conflict lasts.

(d) Conflict lasts.

(e) Second conflict event at zooming ratio $\widetilde{z}_1$.

(f) Labels are free again.

Figure 2.10: Collision between two labels which move with different velocities on the screen when zooming in (or out).

When evaluating the intersection between the dynamic trajectory functions for two labels a, b we get maximally two solutions for an intersection between the two labels' upper/lower borders:

$$x'_{La}(z_c) = x'_{Rb}(z_c) \tag{2.21}$$

$$x_a \cdot d \cdot \frac{1}{z_c - z_a} = x_b \cdot d \cdot \frac{1}{z_c - z_b} + w \tag{2.22}$$

$$x_a \cdot d \cdot (z_c - z_b) = x_b \cdot d \cdot (z_c - z_a) + w \cdot (z_c - z_b) \cdot (z_c - z_a) \tag{2.23}$$

$$\begin{aligned} 0 = {} & z_c^2 \cdot w + z_c \cdot (x_b \cdot d - w \cdot z_b - w \cdot z_a - x_a \cdot d) \\ & - x_b \cdot d \cdot z_a + x_a \cdot d \cdot z_b + w \cdot z_b \cdot z_a \end{aligned} \tag{2.24}$$

$$\begin{aligned} 0 = {} & z_c^2 + z_c \cdot (x_b \cdot \frac{d}{w} - z_b - z_a - x_a \cdot \frac{d}{w}) \\ & + \frac{d}{w} \cdot (x_a \cdot z_b - x_b \cdot z_a) + z_b \cdot z_a \end{aligned} \tag{2.25}$$

$$z_c = -\frac{p}{2} \pm \sqrt{(\frac{p}{2})^2 - q} \tag{2.26}$$

$$\text{with} \quad p = \frac{d}{w} \cdot (x_b - x_a) - z_b - z_a \tag{2.27}$$

$$q = \frac{d}{w} \cdot (x_a \cdot z_b - x_b \cdot z_a) + z_b \cdot z_a \tag{2.28}$$

In equation 2.26, we get two solutions to z_c and accordingly two possible intersections between x'_{La} and x'_{Rb}. We will get another two solutions for the intersection $x'_{Lb}(z_c) = x'_{Ra}(z_c)$ between the two other label borders. This yields four potential conflict events, i.e. start and end points for conflict intervals between two labels.

In the calculations above, we do not accommodate that we did not wanted to regard the part of the projection function left to its singularity (cf. previous section). Hence some of the conflict events could possibly lie on the "wrong side" of the singularity and the labels might not be visible (by equation 2.6) at some intersections of the trajectory functions.

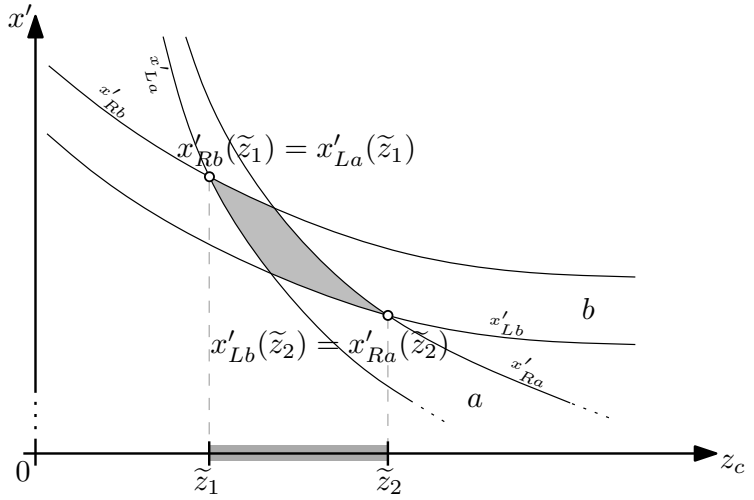


Figure 2.11: Detailed view of two labels' trajectories intersecting. The trajectories of the labels' left and right borders determine the start and end of the conflict interval.

Therefore, some of the solutions in equation 2.26 could be irrelevant. On account of that we will in the following analyse the different possible cases.

We will distinct cases by the labels' x-coordinates x_a and x_b. For **Case 1** we assume both x-coordinates to be positive, in **Case 2** they are negative and in **Case 3** we assume them to have opposite signs. We assume neither $x_a, x_b = 0$ nor $x_a = x_b$ due to the general position of the data points.

Case 1, $x_a > 0$**,** $x_b > 0$**:**

At first we assume that x_a and x_b are both positive. Both trajectory functions have a singularity at z_a and z_b respectively (cf. Section 2.3); and thus will be invalid left of z_a and z_b respectively. As stated in equation 2.6 the labels are displayed for $z_c > z_a$ and $z_c > z_b$ only, i.e. right to the labels' singularities. Intersections (and therefore conflict events) can only occur, if *both* labels are displayed at the same time. That means only conflicts on the right arm of the functions are valid (see Figure 2.12), where $z_c > z_a$ *and* $z_c > z_b$.

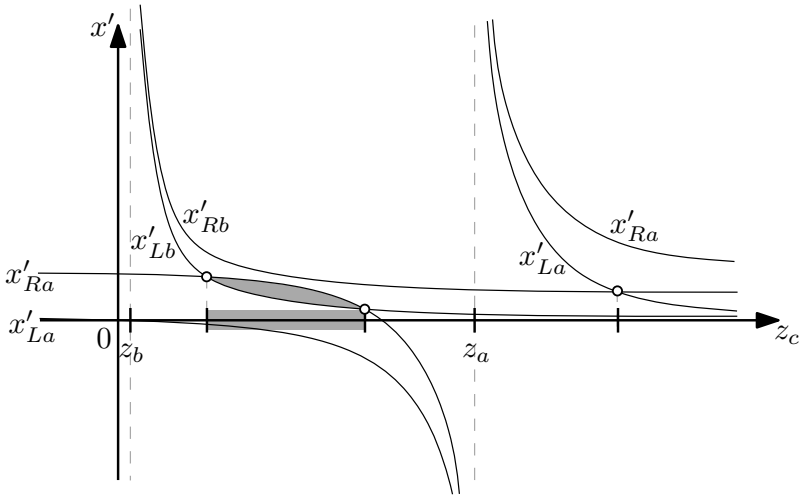


Figure 2.12: Irrelevant conflicts

The first intersection (i.e. with the smallest z-coordinate) between the label borders can only occur at $\widetilde{z}_1 > z_a, z_b$. Label a's borders' trajectories $x'_{La}(z_c)$ and $x'_{Ra}(z_c)$ have their singularity at z_a, label b's borders' trajectories correspondingly at z_b. Due to the assumption of general position, $z_a \neq z_b$; we assume w.l.o.g. that $z_b < z_a$ (for $z_a < z_b$ simply exchange the indices). When the zooming ratio approaches z_a while zooming in (say $z_c > z_a,\ z_c \to z_a$), label a's x'-position increases extremely:

$$x'_a(z_c) \xrightarrow{z_c > z_a,\ z_c \to z_a} +\infty \tag{2.29}$$

while b's x'-position is bounded at z_a (and valid since $z_a > z_b$)

$$x'_b(z_c) \xrightarrow{z_c > z_a,\ z_c \to z_a} x'_b(z_a)\ . \tag{2.30}$$

As b's width w_b is constant and $w_b > 0$, we can assure that

$$\lim_{\substack{z_c > z_a \\ z_c \to z_a}} \Big(x'_{La}(z_c) - x'_{Rb}(z_c) \Big) > 0 \tag{2.31}$$

which means that eventually, label a will be located completely to the right of label b (i.e. in the plots in Figure 2.9 and Figure 2.11, a will be above b). The first intersection at $\widetilde{z}_1$ hence is between label a's left border x'_{La} and label b's right border x'_{Rb}, only if $z_b < z_a$ (between x'_{Lb} and x'_{Ra} otherwise) and the labels will have no conflict for $z_c < \widetilde{z}_1$.

Such a conflict event at $\widetilde{z}_1$ necessarily exists since

$$\lim_{z_c \to +\infty} x'_{La}(z_c) = 0 \tag{2.32}$$

$$\lim_{z_c \to +\infty} x'_{Rb}(z_c) = \lim_{z_c \to +\infty} x'_{Lb}(z_c) + w_b = w_b > 0 \tag{2.33}$$

which means that

$$z_c \to +\infty \Rightarrow x'_{La}(z_c) < x'_{Rb}(z_c) \tag{2.34}$$

and from above we know that

$$z_c > z_a,\ z_c \to z_a \Rightarrow x'_{La}(z_c) > x'_{Rb}(z_c) \tag{2.35}$$

Hence we know that the continuous function

$$x'_{La}(z_c) - x'_{Rb}(z_c) \tag{2.36}$$

has at least one zero in $(z_a, +\infty)$ and therefore the two projection trajectories necessarily intersect. A question that remains is, whether there are any other valid conflict events for the two labels for $z_c > \widetilde{z}_1$, which we will address for $x_b < x_a$ and $x_a < x_b$ separately.

Case 1a, $x_b < x_a$:

If $x_b < x_a$, the conflict event at $\widetilde{z}_1$ will be the only one for the two labels on the whole zooming ratio range. The label borders $x'_{Ra}(z_c)$ and $x'_{Lb}(z_c)$ cannot have any intersection for $z_c > \widetilde{z}_1$:

$$x_a \cdot d\frac{1}{z_c - z_a} + w_a > x_b \cdot d\frac{1}{z_c - z_b} \tag{2.37}$$

since

$$z_a > z_b \tag{2.38}$$

$$z_c - z_b > z_c - z_a \tag{2.39}$$

$$\frac{1}{z_c - z_a} > \frac{1}{z_c - z_b} \tag{2.40}$$

and

$$x_a > x_b \ . \tag{2.41}$$

Equation 2.37 in particular states for $z_c > \widetilde{z}_1$ that $x'_{Ra}(z_c) \neq x'_{Lb}(z_c)$, so that in this case there is no valid intersection between x'_{Ra} and x'_{Lb} at all and thus no additional conflict event due to x'_{Ra} and x'_{Lb}.

There is no additional intersection between the other two borders x'_{La} and x'_{Rb} either. To see this we have a look at the derivatives of the borders' trajectory functions:

$$\frac{x'_{La}(z_c)}{dz_c} = -x_a \cdot d\frac{1}{(z_c - z_a)^2} \qquad \frac{x'_{Rb}(z_c)}{dz_c} = -x_b \cdot d\frac{1}{(z_c - z_b)^2} \tag{2.42}$$

and the same reasoning as above (using equations 2.38 ff.) leads to

$$\frac{x'_{La}(z_c)}{dz_c} < \frac{x'_{Rb}(z_c)}{dz_c} \tag{2.43}$$

for $z_c \in (z_a, +\infty)$, which means that, once one label border "overtook" the other, there is not a second intersection of the two. This means, that there is in total only one conflict event on the whole zooming ratio range $(z_a, +\infty)$.

And furthermore, equation 2.43 shows in particular that

$$\frac{x'_{La}(z_c)}{dz_c}(\widetilde{z}_1) \neq \frac{x'_{Rb}(z_c)}{dz_c}(\widetilde{z}_1) \tag{2.44}$$

which leads to the conclusion, that the trajectories of the label borders do not only touch, but rather truly intersect at $\widetilde{z}_1$. Hence, the conflict event is the starting point of a conflict interval between the two labels.

Summing up, we now know for x_a and x_b being both positive and $x_b < x_a$ (under the assumption $z_b < z_a$), that there is all in all only one conflict event of the labels' borders at $\widetilde{z}_1$. For $z_c < \widetilde{z}_1$, both labels have no conflict with each other. But the conflict event is adjacent to a conflict interval and hence, right to the conflict event starts a conflict interval. Since there is no further conflict event for $z_c > \widetilde{z}_1$, which could mark an end of the conflict interval, we can reason that this conflict event persists for $z_c \to +\infty$.

That means the two labels have a mutual conflict starting at $\widetilde{z}_1$ and persisting for increasing z_c, resulting in the conflict interval $[\widetilde{z}_1, +\infty)$. We denote this type of conflict consisting of a single, unbounded interval by *conflict type (I)*, which is accordingly depicted in the interval diagram in Figure 2.14.

This observation fits well with the intuition, that increasing z_c is equivalent to zooming out and that all labels approach the center of the screen when zooming out very far. So, all labels eventually congregate in the center and therefore overlap. This conflict between the labels does not dissolve when zooming out further.

Case 1b, $x_a < x_b$:

For $z_b < z_a$ the situation becomes way more complex if $x_a < x_b$. Now constellations as illustrated in Figure 2.13 might arise, where in addition to the "smallest" intersection $\widetilde{z}_1$ there are intersections between the trajectories of the labels' borders for $z_c > \widetilde{z}_1$, which leads to additional conflict events.

This situation only occurs, if solutions to $x'_{Lb}(z_c) = x'_{Ra}(z_c)$ exist (see Figure 2.13), i.e. when the value beneath the root in equation 2.26 is positive. This introduces two additional

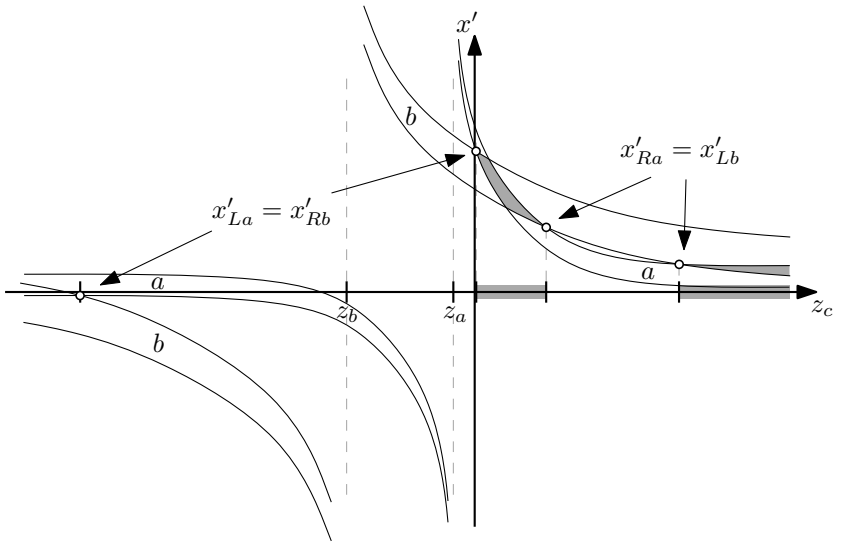


Figure 2.13: Three valid conflict events cause the conflict interval to be split up into two parts.

conflict events $\widetilde{z}_2$ and $\widetilde{z}_3$ for $z_c > \widetilde{z}_1$, which cause the conflict interval in **Case 1a** $[\widetilde{z}_1, +\infty)$ to be split up into two separate conflict intervals $[\widetilde{z}_1, \widetilde{z}_2]$ and $[\widetilde{z}_3, +\infty)$. Opposed to a conflict of type (I), this conflict type consisting of one bounded and one unbounded interval is referred to as *conflict type (II)*, depicted in the conflict diagrams in Figure 2.14.

If the value beneath the root in equation 2.26 was zero, the two labels' borders would only touch and there would be no additional separate conflict interval. This again leads to a conflict of type (I), see Figure 2.14.

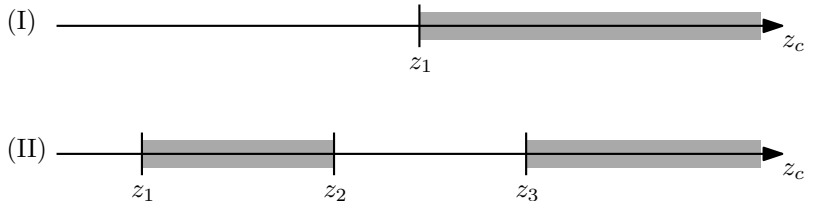


Figure 2.14: Conflict diagrams of the two possible types of conflicts on a 1D screen: (I) One conflict interval $[z_1, +\infty)$ and (II) two conflict intervals $[z_1, z_2]$ and $[z_3, +\infty)$.

Case 2, $x_a < 0$ **and** $x_b < 0$**:**

Equivalently to **Case 1**, we obtain the same types of intersections if x_a and x_b are both negative.

Case 3, $sign(x_a) \neq sign(x_b)$**:**

If $sign(x_a) \neq sign(x_b)$, both arms of the trajectory functions right to the functions' singularities diverge into different directions as depicted in Figure 2.15. With the same argument as above, we will again only consider intersections to the right of both singularities. Due to the offset w of the labels' right borders, we only get one intersection at $\widetilde{z}_1$ between the left border of the label with positive x_p and the right border of the label with negative x_p. This leads to the same situation as in **Case 1a** so that we only have one conflict interval $[\widetilde{z}_1, +\infty)$ and therefore a conflict of type (I) (Figure 2.14).

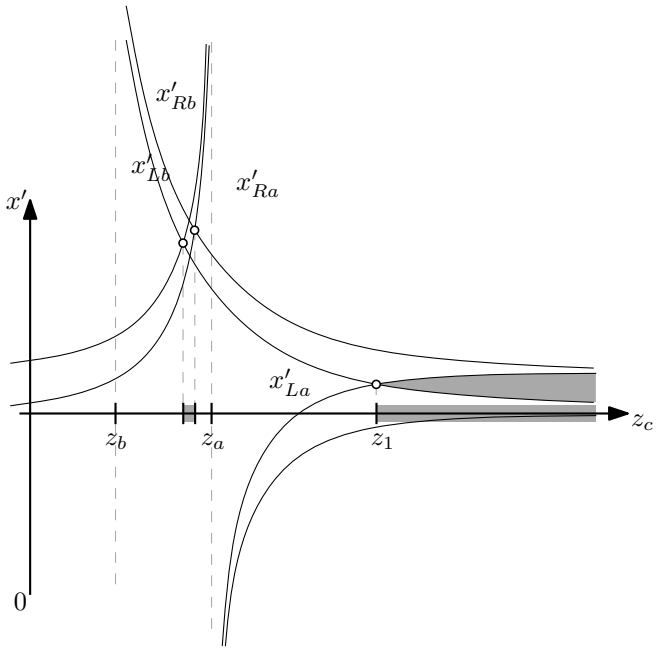


Figure 2.15: Conflicts of labels diverging into different directions. The conflict left to z_a is not a valid conflict.

Using this knowledge, we now can calculate all conflict intervals for each pair of labels being projected onto our one dimensional screen. This results in maximally two conflict intervals per pair. Having n labels, there are maximally $O(n^2)$ possible pairs of labels; hence overall the number of conflict intervals lies in $O(n^2)$.

Lemma 1. *The number of conflict intervals between labels for the 3D fixed position labelling model under zooming lies in $O(n^2)$ on the 1D screen.*

When expanding this model to the 2D screen, we regard conflicts in x- and y-direction separately (see Figure 2.17). Labels then intersect if they intersect on their x and y coordinates simultaneously. Using the notation from equation 2.20 we are searching for the set

$$\begin{aligned}&\{z_c \in \mathbb{R} \mid x'_{La}(z_c) \leq x'_{Rb}(z_c) \wedge x'_{Ra}(z_c) \geq x'_{Lb}(z_c)\}\\ &\cap\ \{z_c \in \mathbb{R} \mid y'_{Ta}(z_c) \leq y'_{Bb}(z_c) \wedge y'_{Ba}(z_c) \geq y'_{Tb}(z_c)\}\ .\end{aligned} \tag{2.45}$$

Taking into account the conflict diagrams presented in Figure 2.14, we combine all types of individual x'- and y'-conflicts. This results in three different types of compound label conflicts (see Figure 2.16) with maximally three conflict intervals per pair. So the number of conflict intervals per pair stays constant and hence we still have $O(n^2)$ intervals in total.

Lemma 2. *The number of conflicts intervals between labels for the 3D fixed position labelling model under zooming lies in $O(n^2)$ on the 2D screen.*

Now we are able to formulate and characterize conflicts between labels. In the next section we will refine this formulation and eventually put the actual label placement problem into terms, i.e. which labels to choose to be rendered for obtaining a good, conflict free labelling.

2.5 Active Range Optimization Problem

As stated earlier, a *conflict interval* between two labels always starts/ends at *conflict events*. Since each label can have at maximum three conflict intervals with any other label, the

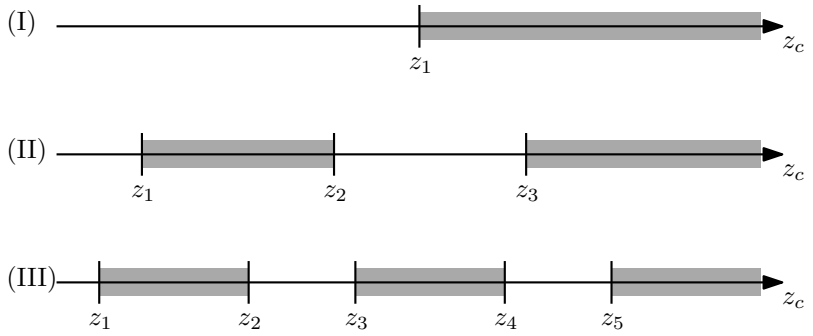
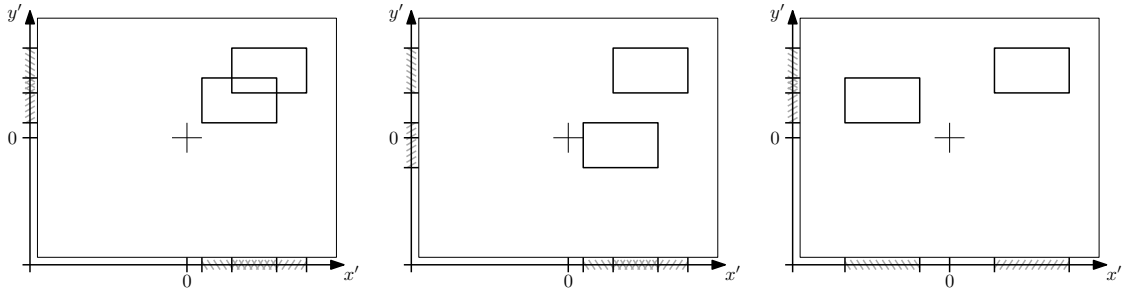


Figure 2.16: Conflict diagrams for 2D label conflicts.



(a) Conflict (b) No conflict (c) No conflict

Figure 2.17: Conflicts on 2D screen only occur, if labels have 1D conflicts in *both* dimensions.

number of active conflict events i.e. starts and ends of conflict intervals lies in $O(n)$ for each label.

Such a conflict interval of two labels a and b bounded by active conflict events $\underline{z}^*_{ab}$ and $\overline{z}^*_{ab}$ can be defined as

$$c^*_{ab} = [\underline{z}^*_{ab}, \overline{z}^*_{ab}], \text{ with } * \in \{1, 2, 3\} \tag{2.46}$$

or

$$c^*_{ab} = [\underline{z}^*_{ab}, +\infty), \text{ with } * \in \{1, 2, 3\} \tag{2.47}$$

for conflict intervals with no upper bound, so that each pair of labels has got up to three conflict intervals in common. But since not every conflict between labels necessarily consists of three conflict intervals, we refer to actually existing conflict intervals by the set C so that a conflict c^*_{ab} exists, only if $c^*_{ab} \in C$.

If a conflict interval exists, its lower and upper bounds are not equal (i.e. it contains more than one element) due to the assumption of general position and since we assume $w_a > 0$ and $w_b > 0$.

$$\forall c^*_{ab} \in C : \underline{z}^*_{ab} < \overline{z}^*_{ab} \tag{2.48}$$

For each pair of labels those intervals are disjoint and sorted in ascending order.

$$\forall c^*_{ab}, c^{*+1}_{ab} \in C : \overline{z}^*_{ab} < \underline{z}^{*+1}_{ab} \tag{2.49}$$

$$\forall c^*_{ab} \in C, * > 1 \Rightarrow c^{*-1}_{ab} \in C \tag{2.50}$$

As stated in the previous section, the first conflict interval begins to the right of both labels' singularities z_a and z_b.

$$\forall c^1_{ab} \in C : \max\{z_a, z_b\} < \underline{z}^1_{ab} \tag{2.51}$$

Yet more strictly we can state that
Lemma 3. $\forall a, b \in S$ *there is a fix* $\varepsilon > 0$ *so that* $\max\{z_a, z_b\} + \varepsilon < \underline{z}^1_{ab}$.

Proof. W.l.o.g. we assume $z_b < z_a$. $\underline{z}^1_{ab}$ exists and is fix with respect to x_a, x_b, y_a, y_b, z_a and z_b. Furthermore, $\underline{z}^1_{ab} > z_a$ so that we choose $\varepsilon = \frac{1}{2}(\underline{z}^1_{ab} - z_a) > 0$. We now can easily show that

$$z_a + \varepsilon = z_a + \frac{1}{2}(\underline{z}^1_{ab} - z_a) \tag{2.52}$$
$$= \frac{1}{2}z_a + \frac{1}{2}\underline{z}^1_{ab} \tag{2.53}$$
$$< \frac{1}{2}\underline{z}^1_{ab} + \frac{1}{2}\underline{z}^1_{ab} \tag{2.54}$$
$$= \underline{z}^1_{ab} \tag{2.55}$$

and therefore

$$\max\{z_a, z_b\} + \varepsilon < \underline{z}^1_{ab} \tag{2.56}$$

□

In the case-by-case analysis above several special situations were left out due to the general position of the data points. However, those special cases can be dealt with by a further in-depth analysis, which we will omit at this point since it would exceed the scope of this thesis and does not lead to any additional insights.

To simplify the analysis in the following, we define the *canonical set of conflict events* to contain only those conflict events that actually mark the start or the end of a conflict interval. That is, if two conflict intervals directly touch (for example $z_2 = z_3$ in Figure 2.16), the two inner conflict events (z_2, z_3) are not in the canonical set. A conflict event in the canonical set is referred to as an *active* conflict event. An active conflict event is always adjacent to exactly one conflict interval.

When now returning to the initial target stated at the end of Section 2.3, we wanted to find a parameterized function taking the current zooming ratio as input and returning a good, valid placement for all labels.

Calculating the labels' positions can be achieved by equations 2.13 ff., and the question was how to prevent labels from overlapping and therefore becoming unreadable. Since we use a fixed position model with only one allowed position per label, the labels' positions are fix relative to their data points. And the data points' trajectories on the screen are given by the data points' coordinates (see equations 2.11 f.) and thus cannot be changed either. Hence there is no point in resolving conflicts by moving labels or data points. To resolve conflicts, we will instead have to remove some of the intersecting labels and display the remaining labels without intersection, so that as much information as possible is accessible to the user.

The actual task of finding a legal labelling thus reduces to finding label placement functions $f_a(z_c)$ $f_a \colon \mathbb{R} \to \{0, 1\}$ for all data points $a \in S$ so that a label is placed at a zooming ratio z_c, only if $f_a(z_c) = 1$. We call this function the *activity function* of a label, since it states whether a certain label a is active (i.e. displayed) at a zooming ratio z_c.

To avoid displaying two labels that overlap at the same time, we have to assure that for each conflict interval $c^*_{ab} \in C$ at maximum one label is displayed:

$$\forall c^*_{ab} \in C : \forall z_c \in c^*_{ab} : f_a(z_c) = 0 \vee f_b(z_c) = 0 \tag{2.57}$$

So for a given zooming ratio z_c, first the activity function for each label is evaluated, and only if the label is supposed to be active for this ratio, the actual position is calculated using the projection functions from equations 2.11 ff., which again take the current zooming ratio z_c as input parameter and return the label's position on the screen. Finally each label is displayed at its on-screen position and all displayed labels are conflict free due to equation 2.57.

The one question remaining is, how to calculate activity functions for the labels that fulfill equation 2.57. This will be the scope of the remainder of this thesis.

We want to present as much information as possible, i.e. present as many labels as possible at each zooming ratio z_c. Hence we are aiming for maximizing the accumulated activity of all labels:

$$\max \sum_{a \in S} \int_{z_c} w(z_c) \cdot f_a(z_c) \tag{2.58}$$

where $w(z_c)$ is a weighting function that allows us to give more importance to different zooming ratio ranges. We call the function to be maximized the *gain function*. The task of optimizing this function will be referred to as the *Active Range Optimization* problem (short ARO) in the following. As we are searching specifically for a solution for the label placement for projected data points $\in \mathbb{R}^3$, we will differentiate this problem from simple 2D labelling ARO, as for example presented in [BNPW10], by calling this problem *ARO-3D*.

In Section 2.4 we introduced the 1D screen for purpose of investigating how to characterize and describe conflicts between labels. This approach of reducing the screen's dimensions has interesting theoretic properties which we will use later on to deduct the theoretical basis for a 2D screen approximate solution. Therefore we will need to distinguish between *ARO-3D on 1D screen* and *ARO-3D on 2D screen*.

In the ARO-3D on 1D screen problem, the screen onto which the data points are projected, is a straight line; the labels are intervals on that line. ARO-3D on 1D screen is not really a proper 3D problem, but rather the projection of 2D data points onto a 1D screen. We still name it *ARO-3D on 1D screen* to distinguish this labelling problem from typical 2D ARO where 2D data points (e.g. a planar map) are "projected" onto a 2D screen and labelled with a 2D labelling algorithm (see [BNPW10],[GNR11]).

We now have introduced the basic mathematical tools to develop a first solution to the ARO-3D problem. In the next chapter we will propose a set of rules for choosing which labels to be displayed to gain a good labelling. We will suggest different approaches to solve ARO-3D according to these rules.

3. Labelling Model A

In this chapter we use the previously defined mathematical descriptions to derive a first model for labelling 3D data points. For this purpose, we first introduce criteria for a good labelling and how this can be formalized using the above definitions. The arising formal problem is NP-hard, which we prove in Section 3.2. Due to the NP-hardness, we introduce and analyse several MILP formulations and heuristics, which are evaluated quantitatively in the last section of this chapter.

3.1 Labelling Criteria & Optimization Goals

As already stated in the previous chapter, we want to present as much information as possible to the user. This means, we are aiming for maximizing the total number of present labels for all zooming ratios, which is in terms of equation 2.58:

$$\max \sum_{a \in S} \int_{z_c} w(z_c) \cdot f_a(z_c) \ . \tag{3.1}$$

We assume that the weighting function $w(z_c)$ is constant for the whole zooming ratio range. Another possible weighting function would be $\frac{1}{z}$ or $\log(z)$ for example for dense, centered data. However, we stay with the constant weighting function for the rest of this work. Investigations on optimizing other gain functions is left open for future work.

We can easily see that, if the gain function is integrated over $f_a(z_c)$ for $z_c \in (-\infty, \infty)$, the solution is unbounded since all label projection functions are defined for $z_c \to \infty$:

$$z_c \to \infty > z_a \qquad \forall a \in S \tag{3.2}$$

and therefore at least one label can be active for $z_c \to \infty$. Hence, at least one label placement function $f_a(z_c)$ would be non-zero and the integral in equation 3.1 becomes unbounded.

To prevent this, we bound the zooming ratio parameter by an interval:

$$z_c \in [\underline{z}_c, \overline{z}_c] \tag{3.3}$$

and gain function to be maximized is

$$\max \sum_{a \in S} \int_{\underline{z}_c}^{\overline{z}_c} f_a(z_c) dz_c \ . \tag{3.4}$$

In every application that is based on user interaction, the zooming ratio range is bound to be finite. This completely aligns with the above constriction.

In general the size of the screen is bounded too in real world application. But we will not include that constraint at this point since the amount contributed by a label a leaving the screen is strictly bounded bound by z_a, see equation 2.6. In Section 4.3 however, we will incorporate the boundedness of the screen into the model to obtain an approximate solution.

Simply optimizing function 3.1 is not enough as we want to gain a "good" labelling. The overall target when presenting data to the user is the usability, in this case the readability of the information presented by the labels. Several criteria for a good labelling were introduced by Been et al. [BDY06].

The authors make a basic observation leading to different labelling constraints. If the labelling layout rapidly changes when the user interacts with the scene this strongly reduces the usability of the labelling layout. On a volatile user interface it is very hard for the user to track data points or retrieve information when the whole screen is blinking and flickering with labels.

One of the criteria introduced by Been et al. [BDY06] prohibits "jumping" of labels. If labels are allowed to take different positions relative to the data point's position, the label should not jump between the different positions. As our labelling model only allows one position relative to the data point for each label, the labels cannot possibly jump.

Another constraint, which also aims to avoid disturbing the user with flickering labels, is based on the observation that labels with a relatively short activity seem to flicker on the screen. Labels popping up and vanishing at each conflict event rather reduce the readability than provide any information.

To account for this, the labels' activity functions use a single, unsplittable *activity interval* per label. That means we can switch on a label at any zooming ratio for which it is not blocked by another active label (i.e. start of the activity interval) and once it is switched off again (end of the activity interval), we may not activate it a second time.

Using the notation from above, we are searching for an *activity interval*, rather than an activity function, that defines for which zooming ratios the label is visible on the screen

$$[\underline{z}_a, \overline{z}_a] \subset [\underline{z}_c, \overline{z}_c] \colon f_a(z_c) = 1 \Leftrightarrow z_c \in [\underline{z}_a, \overline{z}_a] \tag{3.5}$$

for each label a in S so that no two activity intervals overlap within a conflict interval:

$$\forall c^*_{ab} \in C : ([\underline{z}_a, \overline{z}_a] \cap c^*_{ab}) \cap ([\underline{z}_b, \overline{z}_b] \cap c^*_{ab}) = \emptyset \tag{3.6}$$

The gain function now can easily be expressed by

$$\max \sum_{a \in S} \overline{z}_a - \underline{z}_a \ . \tag{3.7}$$

This almost directly leads to a first MILP formulation to solve the ARO-3D labelling problem. But before actually presenting a MILP formulation in Section 3.3, we further investigate on the computational complexity of this newly arisen problem in the following section.

3.2 NP-hardness

In [BNPW10] Been et al. present a dynamic labelling model very similar to the one developed in this work. They investigate on active range optimization for dynamic point

labelling on a zoomable 2D map. In particular, they show NP-completeness of *Simple 2d-ARO with proportional dilation.*

"Simple" 2d-ARO denotes active range optimization where labels can be activated on a single, unsplittable activity interval located anywhere within an interval of possible zooming ratios. The zooming ratio is expressed by means of a scaling factor s, which denotes the ratio of world distances to on-screen distances. That means having a distance δx in world coordinates and its corresponding on-screen distance $\delta x'$ (which could be for example the label's on-screen width or height) the scaling factor is defined as

$$s = \frac{\delta x}{\delta x'} \ . \tag{3.8}$$

Since maps are usually presented strongly demagnified on screen, the map scale is $s \gg 1$ in most mapping applications.

To show the relation between the labelling model presented by Been et al. and our model, we will embed an arbitrary 2d-ARO instance in our ARO-3D labelling model by placing all 2D data points on a plane in 3D at $z = 0$. The plane is parallel to the viewplane and the world x- and y-coordinates of a 2D data point in the 2d-ARO model are the same as the corresponding x- and y-coordinates in our model.

Considering the definition of the zooming ratio z_c as the camera's position in this work, the distance $\delta x = x_b - x_a$ between two points in world coordinates is by equations 2.2 ff. (with $z_a = z_b = 0$)

$$\frac{\delta x}{\delta x'} = \frac{z_c}{d} \ . \tag{3.9}$$

That means the delation component defined by Been et al. is of the form

$$D^L(s) = bs \qquad , b = \frac{1}{d} > 0 \ , \tag{3.10}$$

which basically reflects the property, that the labels on-screen size is fix for any zooming ratio z_c or scaling factor s. Therefore when embedding Simple 2d-ARO instance with proportional delation into the ARO-3D model as stated above, we obtain an equivalent dynamic labelling model instance.

Since the solution to an ARO-3D problem respects the same constraints as the necessary for a solution to the corresponding 2d-ARO problem, the ARO-3D on 2D screen optimization problem reduced to a plane at $z = 0$ is equivalent to the Simple 2d-ARO with proportional dilation problem.

Now it is obvious that, when Simple 2d-ARO with proportional dilation is NP-hard, so is ARO-3D on 2D screen.

Theorem 1. *ARO-3D on 2D screen problem is NP-hard. More precisely, given a set $S \subset \mathbb{R}^3$ of data points in 3D and associated axis aligned labels, it is NP-hard to decide, whether there is a set of activity intervals $\{[\underline{z}_a, \overline{z}_a] \mid a \in S\}$ so that active labels are conflict free and $\sum_{a \in S} \overline{z}_a - \underline{z}_a > K$ for a given real number K.*

3.3 MILP Formulation

As previously noted in Section 3.1 the problem formulation of ARO-3D in Labelling Model A itself already leads to a basic MILP formulation. The optimal solution to ARO-3D is the set of activity intervals

$$\{[\underline{z}_a, \overline{z}_a] \mid a \in S\} \tag{3.11}$$

which maximizes the gain function (equation 3.4)

$$\sum_{a \in S} \overline{z}_a - \underline{z}_a \tag{3.12}$$

while no two activity intervals should overlap within a mutual conflict interval

$$\forall c^*_{ab} \in C : ([\underline{z}_a, \overline{z}_a] \cap c^*_{ab}) \cap ([\underline{z}_b, \overline{z}_b] \cap c^*_{ab}) = \emptyset \ . \tag{3.13}$$

This constraint on the labels' conflict can be rewritten to

$$\forall c^*_{ab} \in C : \ \underline{z}_a > \overline{z}^*_{ab} \ \vee \ \underline{z}_b > \overline{z}^*_{ab} \ \vee \ \overline{z}_a < \underline{z}^*_{ab} \ \vee \ \overline{z}_b < \underline{z}^*_{ab} \tag{3.14}$$

$$\vee \ \overline{z}_a < \underline{z}_b \ \vee \ \overline{z}_b < \underline{z}_a \ . \tag{3.15}$$

The first line (3.14) obviously resolves the conflict by completely excluding one of the two labels from this conflict interval. The second line (3.15) allows both labels to be active on a part of this conflict interval, but only if the two activity intervals do not overlap, i.e. if one of them lies completely left or right of the other.

We translate this disjunctive expression into a valid MILP formulation by applying the *big-M technique* [GNS08]. For this purpose we have to introduce additional binary variables, one for each clause of the disjunction. A binary variable b_k ($k = 1..6$) is *true*, i.e. $b_k = 1$, if the corresponding condition in the disjunction is violated. Since one satisfied condition is enough to resolve the conflict, five out of six binary variables may be true for a valid solution.

$$\forall c^*_{ab} \in C : \ \underline{z}_a > \overline{z}^*_{ab} - b_1 \cdot M \tag{3.16}$$

$$\underline{z}_b > \overline{z}^*_{ab} - b_2 \cdot M \tag{3.17}$$

$$\overline{z}_a < \underline{z}^*_{ab} + b_3 \cdot M \tag{3.18}$$

$$\overline{z}_b < \underline{z}^*_{ab} + b_4 \cdot M \tag{3.19}$$

$$\overline{z}_a < \underline{z}_b + b_5 \cdot M \tag{3.20}$$

$$\overline{z}_b < \underline{z}_a + b_6 \cdot M \tag{3.21}$$

$$\sum_{k=1}^{6} b_k \leq 5 \tag{3.22}$$

$$b_k \in \{0, 1\}, k = 1..6 \tag{3.23}$$

with M being "big enough". Using these constraints, we propose the first MILP formulation:

MILP 1

$$\text{maximize} \sum_{a \in S} \overline{z}_a - \underline{z}_a \tag{3.24}$$

$$\text{s.t. } \forall a \in S: \tag{3.25}$$

$$\underline{z}_a < \overline{z}_a \tag{3.26}$$

$$\max\{\underline{z}_c, z_a\} < \underline{z}_a, \overline{z}_a \tag{3.27}$$

$$\underline{z}_a, \overline{z}_a < \overline{z}_c \tag{3.28}$$

$$\forall c^*_{ab} \in C: \tag{3.29}$$

$$\underline{z}_a > \overline{z}^*_{ab} - b_1 \cdot M \tag{3.30}$$

$$\underline{z}_b > \overline{z}^*_{ab} - b_2 \cdot M \tag{3.31}$$

$$\overline{z}_a < \underline{z}^*_{ab} + b_3 \cdot M \tag{3.32}$$

$$\overline{z}_b < \underline{z}^*_{ab} + b_4 \cdot M \tag{3.33}$$

$$\overline{z}_a < \underline{z}_b + b_5 \cdot M \tag{3.34}$$

$$\overline{z}_b < \underline{z}_a + b_6 \cdot M \tag{3.35}$$

$$\sum_{k=1}^{6} b_k \leq 5 \tag{3.36}$$

$$b_k \in \{0,1\}, k = 1..6 \tag{3.37}$$

If $\underline{z}_a = \overline{z}_a$, we assume the corresponding label a to be inactive for the whole zooming ratio interval. Its contribution to gain function is zero.

Additionally, to produce a valid solution and maintain solvability, we have to substitute $\underline{z}^*_{ab}$ and $\overline{z}^*_{ab}$ by corrected conflict interval borders:

$$\underline{z}^*_{ab} := \max\{\underline{z}^*_{ab}, \underline{z}_c\}, \qquad \overline{z}^*_{ab} := \min\{\overline{z}^*_{ab}, \overline{z}_c\}\ . \tag{3.38}$$

This means that conflict intervals "stop" at the minimum and maximum zooming ratio. That does not actually alter the value of the optimal solution to MILP 1, but maintains solvability of the MILP, if labels cannot be activated at all.

The number of continuous variables in MILP 1 is $2n$, with n being the number of labels in S. The number of binary variables is $6m$ with m being the number of conflicts. Since the number of conflicts is in $O(n^2)$, so is the number of binary variables of MILP 1.

Even for fairly small instances, MILP 1 takes extremely long to compute the optimal solution, see Section 3.6 for further details.

3.4 Discretization and ILP Formulations

In this section we construct a purely combinatorial ILP model, avoiding continuous variables like in the MILP above.

It seems likely that it will suffice for each label to consider only its own conflict events as potential boundaries for each label's activity interval. In this case, the labels activity interval would always start and end at the beginning of a conflict event between this label and another.

We assume in the optimal solution exists a label whose activity starts "somewhere" between two conflict events. If this label has no conflict with any other label at this position,

it is safe to extend this activity interval towards the next conflict event. No additional conflicts are added and the overall objective increases. This would be a contradiction to the optimality of the solution, so that this sort of situation cannot happen.

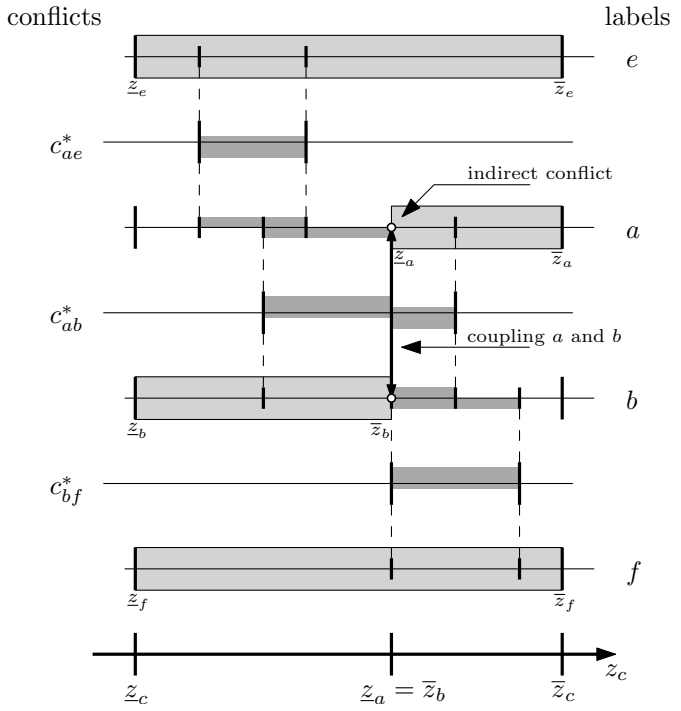


Figure 3.1: Activity intervals of labels a and b are coupled at $\underline{z}_a = \overline{z}_b$ since otherwise loss would occur. Label a's activity interval ends at an indirect conflict event of b's.

However, the problematic case occurs, if the assumed interim-end of the label's activity interval occurs within a conflict with another label as depicted in Figure 3.1. Both labels are active within their mutual conflict interval, but do not overlap. This situation corresponds to equation 3.34 and 3.35 in MILP 1, so that $b_5 = 0$ or $b_6 = 0$. When moving one label's activity interval border to coincide with (any) conflict event, the other one has to "follow" to make up for this loss. But both labels are blocked by other conflicts and therefore are not free to follow. A deadlock occurs so that at least one of the labels' activity intervals ends at an *indirect* conflict event.

Figure 3.2 shows an instance of ARO-3D on 1D screen in which such a deadlock is constructed with actual label trajectories.

Hence, for a combinatorial solution, we cannot only consider activity interval borders, that coincide with the label's local conflict events. We rather have to take into account all conflict events of all labels. As indicated in equations 3.38 f., we consider the zooming ratio interval bounds to be conflict events too for the following lemma.

Lemma 4. *There always exists an optimal solution to ARO-3D in Labelling Model A that uses activity intervals that start and end at conflict events only.*

Proof. Lets assume we have an optimal solution to ARO-3D in Labelling Model A with a non-empty activity interval that starts "somewhere" in between two conflict events. Therefore there has to be one conflict event left to the activity interval border and a second one right to it.

If the activity interval completely lies between the two conflict events, non of the two is contained in the activity interval, otherwise exactly one of the conflict events is contained

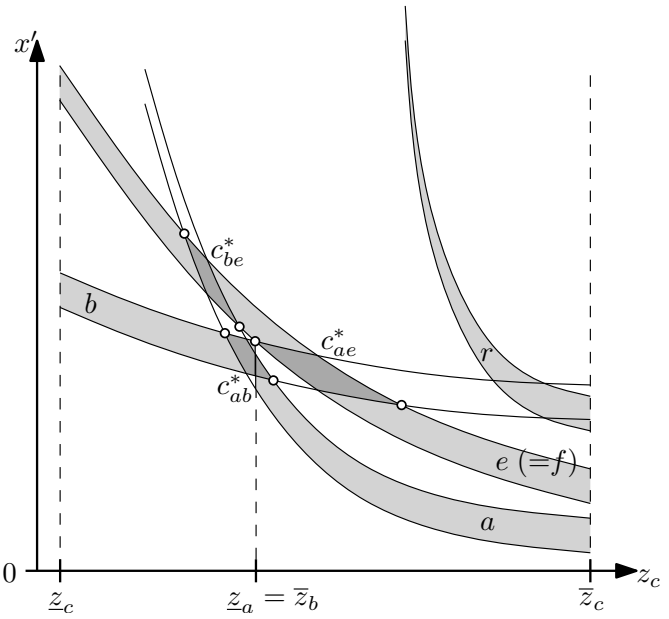


Figure 3.2: Example trajectories of an ARO-3D on 1D screen instance with indirect conflicts. The additional label r is necessary to force the optimal solution into this configuration.

within the activity interval. Both conflict events cannot be contained in the activity interval, because then the activity would not stop in between those events.

At first we assume the corresponding label to have no conflict with another label between the two conflict events. Then we can easily extend the activity interval towards the conflict event that's not contained within the activity interval without introducing a conflict with another label, see Figure 3.3 This would result in an increase of the gain function which contradicts the assumption on optimality.

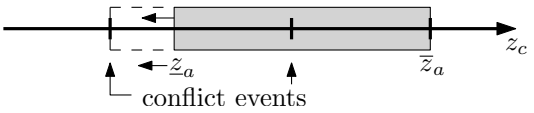
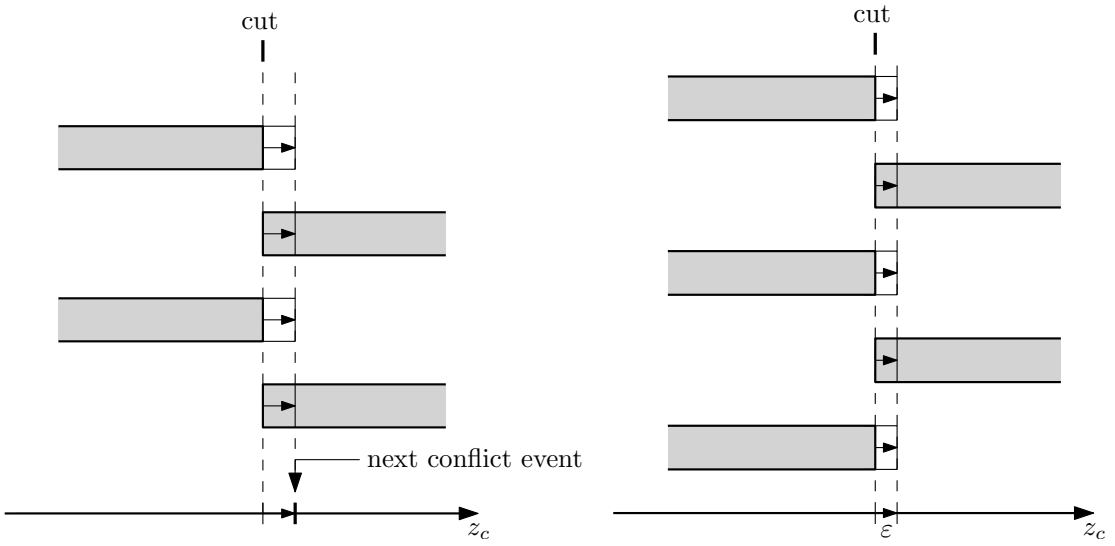


Figure 3.3: Extending an unblocked activity interval towards the next conflict event.

The second case to be considered is the label having active conflicts with one or more other labels between the two conflict events which hinder the label from being extended towards the conflict event not contained in the interval (cp. Figure 3.4).

Since there is no additional conflict event, labels either have a conflict for the whole interval, or non at all. Labels only hinder each other from extending, when they actually have a conflict. We then have a "cut" between two conflicting labels and one of them is left, the other one right of it. If there are multiple different such cuts, we start with leftmost and introduce virtual conflict event at next cut right to it. This results in one left group of labels and one right group of labels, as depicted in Figure 3.4.

If the number of labels to the right and to the left of this cut are equal (Figure 3.4(a)), we move cut (i.e. all activity interval borders starting/ending there) to one side. The gain of one group makes up for loss of the other group. It is legal to move the cut, since the conflict relation does only change at (virtual) conflict events.



(a) $2\varepsilon - 2\varepsilon = 0$ (b) $3\varepsilon - 2\varepsilon = \varepsilon > 0$

Figure 3.4: Moving the cut either preserves the gain value or increases it.

If the number of labels in the two groups were not equal (Figure 3.4(b)), we move the cut towards the side of the group with less label by an $\varepsilon > 0$, which increases the gain function but does not introduce additional conflicts. This is a contradiction to the optimality.

Applying this reasoning to all cuts in the optimal solution, we construct a valid, cut-free solution from optimal solution without decreasing gain function. □

That means we can represent activity intervals completely by use of conflict events. Since number of overall conflicts is in $O(n^2)$ and each conflict contains two conflict events, there are $O(n^2)$ conflict events all in all. As an activity interval of a label starts and ends at conflict events, there are $O(n^4)$ possible combinations of start and end point for an activity interval. Testing all possible activity intervals for a label therefore is equal to testing all $O(n^4)$ combinations of two conflict events.

In the first ILP we use one variable per possible activity interval per label. Since there are $O(n^4)$ possible activity intervals per label and n labels this yields in total $O(n^5)$ binary variables. Additionally we add constraints which mutually exclude conflicting activity intervals of every two labels having a conflict.

To simplify notation in the following, we formulate the canonical set of all active conflict events

$$E_C = \{\underline{z}^*_{ab}, \overline{z}^*_{ab} \mid c^*_{ab} \in C\} \ , \tag{3.39}$$

which we assume to be sorted ascending by z-coordinates so that

$$\forall z_k, z_{k+1} \in E_C : z_k \leq z_{k+1} \ . \tag{3.40}$$

We use this set of conflict events to formulate the set P_a of all possible activity intervals for a label a

$$P_a = \{\pi_a = [\underline{z}_a, \overline{z}_a] \mid \underline{z}_a, \overline{z}_a \in E_C\} \ . \tag{3.41}$$

For each possible activity interval for a label a we add one binary variable to the ILP indicating that label a is active on activity interval π_a

$$f_{\pi_a} \in \{0, 1\} \quad , \pi_a \in P_a \tag{3.42}$$

Each activity interval contributes a value to the gain function that is equal to the interval's length

$$w_{\pi_a} = \overline{z}_a - \underline{z}_a \quad , [\underline{z}_a, \overline{z}_a] = \pi_a \in P_a \tag{3.43}$$

ILP 1

$$\text{maximize} \sum_{a \in S} \sum_{\pi_a \in P_a} w_{\pi_a} \cdot f_{\pi_a} \tag{3.44}$$

$$\text{s.t. } \forall a \in S : \tag{3.45}$$

$$\sum_{\pi_a \in P_a} f_{\pi_a} \leq 1$$

$$\forall c^*_{ab} \in C \ : \ \forall c_k, c_{k+1} \in E_C \ , \ [c_k, c_{k+1}] \cap c^*_{ab} \neq \emptyset \ : \tag{3.46}$$

$$\sum_{\substack{\pi_a \in P_a \\ [c_k, c_k+1] \cap \pi_a \neq \emptyset}} f_{\pi_a} + \sum_{\substack{\pi_b \in P_b \\ [c_k, c_k+1] \cap \pi_b \neq \emptyset}} f_{\pi_b} \leq 1$$

$$\forall a \in S \, \forall \pi_a \in P_a : \tag{3.47}$$

$$f_{\pi_a} \in \{0, 1\}$$

Line 3.45 assures that maximally one activity interval per label is active and therefore contributes to the gain function in line 3.44. Line 3.46 introduces for every two neighbouring conflict events within a conflict interval a constraint that hinders the two labels from being activated at the same time within the conflict interval. Since there are $O(n^2)$ conflicts which can contain $O(n^2)$ conflict events each, this yields $O(n^4)$ constraints.

The huge number of variables of this ILP increases the running time. To improve on this, we introduce a second ILP with reduced number of binary variables in the following.

The idea for the reduction is based on the observation that an activity interval basically consists of a series of conflict events and segments in between. Instead of representing each possible activity interval of a label by an individual binary variable, we introduce a binary variable for each segment, which states whether this segment is part of the activity interval. As an activity interval is defined by its start and end conflict event, all the conflict events in between actually have no potential influence.

This gives rise to the idea to couple the state of successive active segments and determine them at both ends by a start event and an end event. We achieve this by means of binary chaining variables between every two neighboured active segments.

The set of segments of activity intervals for a label a is denoted by S_a so that

$$S_a = \{\sigma_a = [z_k, z_{k+1}] \ \mid \ z_k, z_{k+1} \in E_C\} \ , \tag{3.48}$$

which is used for referring to the binary variables as

$$f_{\sigma_a} \in \{0, 1\} \quad , \sigma_a \in S_a \tag{3.49}$$

and each segments gain value as

$$w_{\sigma_a} = z_{k+1} - z_k \quad , [z_k, z_{k+1}] = \sigma_a \in S_a \tag{3.50}$$

Conflicts between two labels' trajectories are simply modeled by the corresponding segments being mutually excluded.

$$\forall c^*_{ab} \in C \ : \ \forall c_k, c_{k+1} \in E_C \ , \ [c_k, c_{k+1}] \cap c^*_{ab} \neq \emptyset \ :$$
$$f_{\sigma_a} + f_{\sigma_b} \leq 1 \quad , \ \sigma_a = \sigma_b = [c_k, c_{k+1}] \tag{3.51}$$

Additionally we introduce binary helper variables, which are used to "trigger" the start of an activity interval. This trigger variable marks the transition between $f_{\sigma_a} = 0$ and $f_{\sigma_a+1} = 1$.

$$t_{\sigma_a} \in \{0,1\} \quad , \sigma_a \in S_a \tag{3.52}$$
$$f_{\sigma_a} + t_{\sigma_a} \geq f_{\sigma_a+1} \tag{3.53}$$

Only one of the t_{σ_a} per label a is allowed to be true as there is only one unique, unsplittable activity interval and therefore only one start of an activity interval per label.

$$\forall a \in S : \sum_{\sigma_a \in S_a} t_{\sigma_a} \leq 1$$

ILP 2

$$\text{maximize} \sum_{a \in S} \sum_{\sigma_a \in S_a} w_{\sigma_a} \cdot f_{\sigma_a} \tag{3.54}$$
$$\text{s.t. } \forall a \in S : \tag{3.55}$$
$$f_{\sigma_a} + t_{\sigma_a} \geq f_{\sigma_a+1} \quad , \sigma_a, \sigma_a + 1 \in S_a$$
$$\sum_{\sigma_a \in S_a} t_{\sigma_a} \leq 1$$
$$\forall c^*_{ab} \in C \ : \ \forall c_k, c_{k+1} \in E_C \ , \ [c_k, c_{k+1}] \cap c^*_{ab} \neq \emptyset \ : \tag{3.56}$$
$$f_{\sigma_a} + f_{\sigma_b} \leq 1 \quad , \sigma_a = \sigma_b = [c_k, c_{k+1}]$$
$$\forall a \in S \, \forall \sigma_a \in S_a : \tag{3.57}$$
$$f_{\sigma_a}, t_{\sigma_a} \in \{0,1\}$$

This solution decreases the number of variables to $O(n^3)$ but surprisingly increases the running time necessary to find the optimal solution (see Section 3.6).

3.5 Heuristics

In this section we present heuristic approaches to circumvent the high computation time necessary to solve the (M)ILP formulations in the previous section. The first two solutions are still ILP formulations, but with fewer variables and constraints, which reduces the computational burden but still leads to a certain degree of optimality and we expect them to yield results not too far away from the optimal solution. For the remaining heuristics we give no guarantee on the quality of the computed solution. All solutions are evaluated in experiments presented in Section 3.6.

ILP1a & ILP2a heuristics

As already stated in Lemma 4, we have to take into account all conflict events as potential starts and ends of the labels' activity intervals. This leads to a huge number of binary variables in the combinatorial approaches of ILP1 and ILP2. However, in exemplary tests we observed that indirect conflicts have a very small influence on the gain value of the optimal solution. Bad situations like the one depicted in Figure 3.1 seem to occur rarely and lead to minor losses if not accounted for.

This gives rise to the idea to ignore these special cases and only consider activity interval starts (and ends) for a label that coincide with the label's conflict events and not with the

conflict events of all other labels. The resulting altered ILP formulations are structurally exactly the same except for the reduction of binary variables.

We refer to the altered version of ILP1 by **ILP1a** and similarly to the altered version of ILP2 by **ILP2a**.

Since each label has maximally $O(n)$ conflicts with other labels, the number of possible different activity intervals per label is reduced from $O(n^4)$ to $O(n^2)$. Therefore, the total number of binary variables for ILP1a is now in $O(n^3)$, which is quite an improvement compared to $O(n^5)$ binary variables in ILP1.

Likewise, we can observe a big improvement for ILP2 and ILP2a. Since the number of activity interval segments is linear in the number of conflict events, we obtain $O(n)$ binary variables per label in ILP2a and therefore $O(n^2)$ binary variables in total for ILP2a.

The error due to the simplification is for both heuristic ILP formulations the same, as both are based on the exclusion of the same special cases. The reduction in the number of binary variables leads to a high reduction in computation time compared to the original ILP formulations, as we will show in the experimental evaluation in Section 3.6.

Greedy heuristics

Additionally to the previously presented ILP heuristics, we implemented two greedy heuristics, which are intended to way faster than the (sub-)optimal ILP formulations.

For the greedy heuristics, we iterate over all available labels and place their activity interval one by one greedily. We activate each label on the biggest possible zooming ratio interval that we can choose without introducing conflicts with other already placed labels. If there exists no conflict free activity interval for that label, it is simply discarded and we proceed with the next label.

We implemented two versions of this heuristic that only differ in the order, in which labels are chosen to be placed. In the first version, which we will refer to as **GRand**, we randomly choose the next label from the set of labels remaining to be activated. In **GZOrder**, we sort the labels descending by the z-coordinate of their data point and iterate in this order to place the labels. The idea behind this second approach is, that labels that are closer to the camera should be preferred over labels very far away from the camera.

3.6 Quantitative Evaluation

The intention of this experimental evaluation is to give a rough insight on the performance of the previously presented algorithms rather than to provide an in-depth analysis.

We implemented all previously described algorithms in C++ using the Gurobi 5.6 optimizer to solve the (M)ILPs. Gurobi used four threads for solving the MILP and ILPs while the rest of the implementation is purely sequential and not highly optimized. The test system had 48 CPUs, each with 2.1GHz clock frequency, and 256GB main memory in total.

We tested our algorithms on instances with n data points placed uniformly at random within a cube of 20 by 20 by 20 units, so that a data point's coordinate (x, y, z) is in $[-10, 10]^3$. The zooming ratio interval is $[0, 10]$ and the viewplane's distance $d = 1$. All labels are unit squares with side length $w = h = 0.2$.

In a first experiment, we compare the running time of the three optimal algorithms MILP, ILP1 and ILP2. Gurobi is preempted after ten minutes for each run, leading to suboptimal solutions.

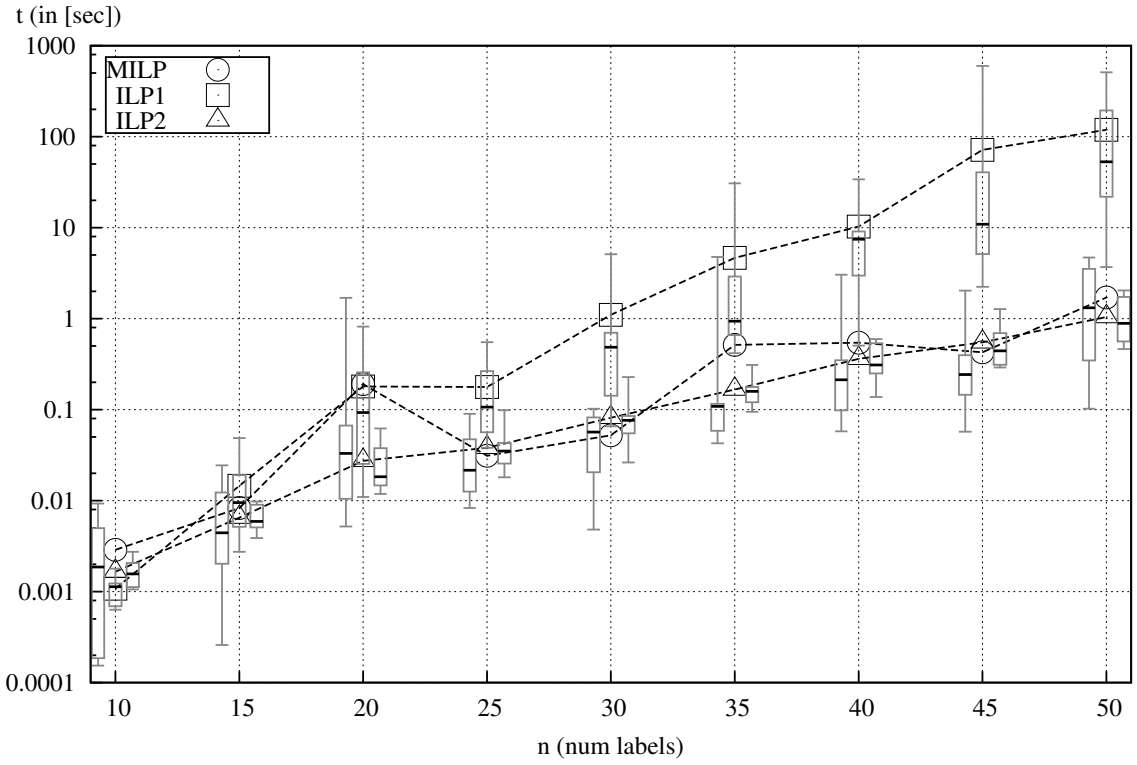


Figure 3.5: (M)ILP running time comparison, displayed on logarithmic scale. The boxplots are sorted within the groups from left to right: MILP, ILP1, ILP2.

In Figure 3.5 we show the results of this test run. We performed the tests with increasing number n of labels from 10 to 50 with step size 5. For each n, we show the mean running time to solve instances of this size (dashed lines) as well as robust location and dispersion measures which are represented as boxplots (sorted in groups from left to right: MILP, ILP1 and ILP2).

For very few labels, these solutions already take much computation time. The purely combinatorial ILP1 performs worse than the MILP formulation. In some test runs, we observed that ILP1 consumed a fairly large amount of memory (>32 GB for $n = 40$ labels). ILP2, with $O(n^3)$ binary variables instead of $O(n^5)$ in ILP1, is apparently faster than the other two optimal algorithms, but still rather slow.

We conducted a second experiment to evaluate the two ILP heuristics ILP1a and ILP2a. We use ILP2 to compute the optimal solution to compare with and chose n between 10 and 150, increasing with step size 10. The results can be seen in Figure 3.6, again mean running time and dispersion measures (boxplots in groups from left to right: ILP2, ILP1a, ILP2a). The heuristic ILP formulations are faster than the optimal ILP2. Surprisingly, now ILP1a performs better than ILP2a although it has a higher number of binary variables.

We also compared the objective value of the ILP heuristics with the optimal solution, which is shown in Figure 3.7. As already mentioned earlier, despite their sub-optimality, the objective value of these two heuristics is very close to the optimum.

In a third series of experiments, we compared the greedy heuristics with the solutions of ILP1a. Although ILP1a is not optimal, it is very close to the optimum and the high number of labels in these tests forbids to calculate the optimal solution with ILP2 or MILP. Both greedy heuristics compete very well against ILP1a in terms of running time, as can

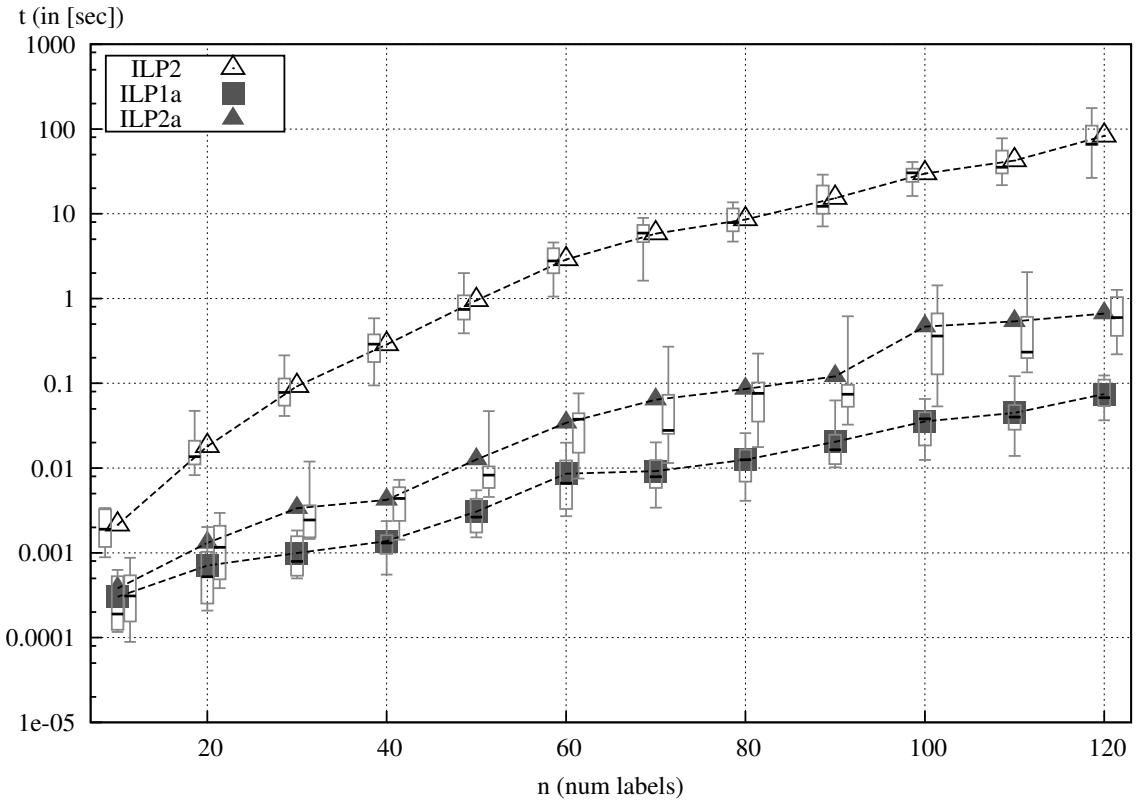


Figure 3.6: ILP heuristics running time comparison, displayed on logarithmic scale. The boxplots are sorted within the groups from left to right: ILP2, ILP1a, ILP2a.

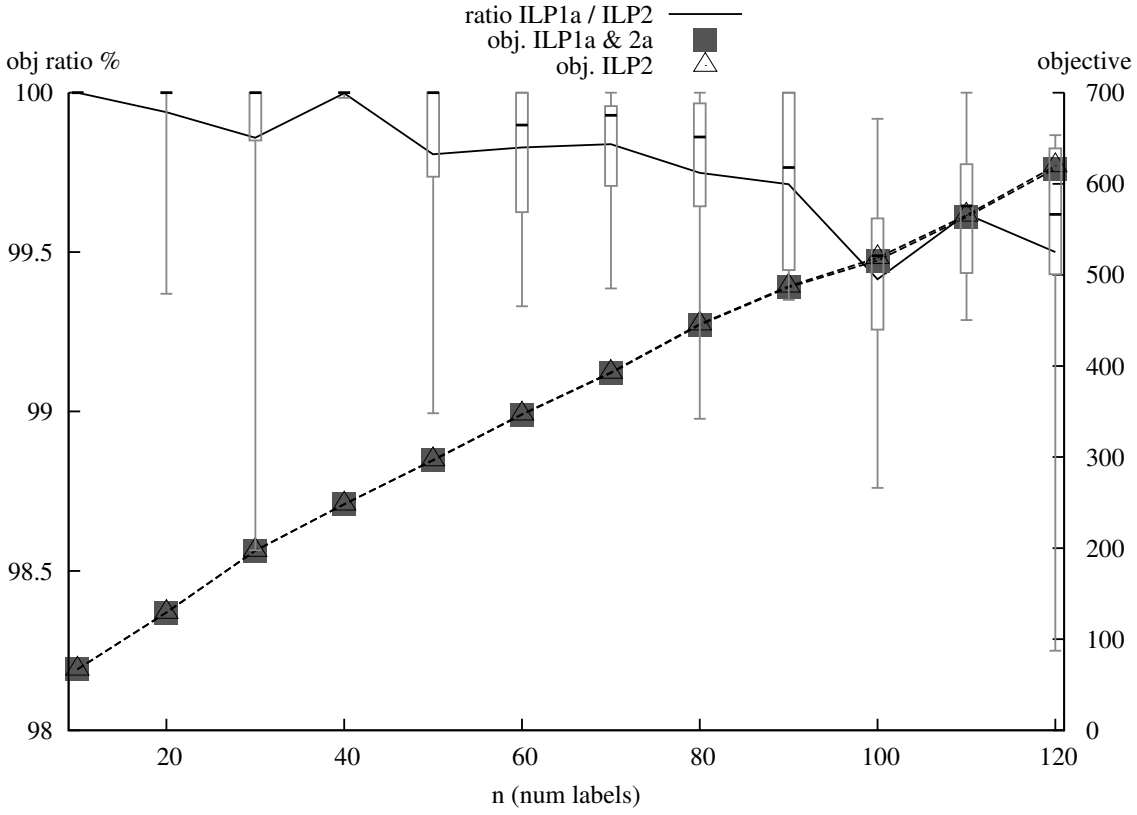


Figure 3.7: ILP heuristics objective value comparison.

be seen in Figure 3.8. Even for a high number of labels, the computation time remains at about one second.

The two greedy heuristics also compete relatively well in terms of the objective value of the computed solutions, which is depicted in Figure 3.9. Recall that the objective value of ILP1a, to which the two are compared, is not optimal but close to the optimum. It is also noteworthy, that the randomized heuristic GRand performs better than the sorted GZOrder.

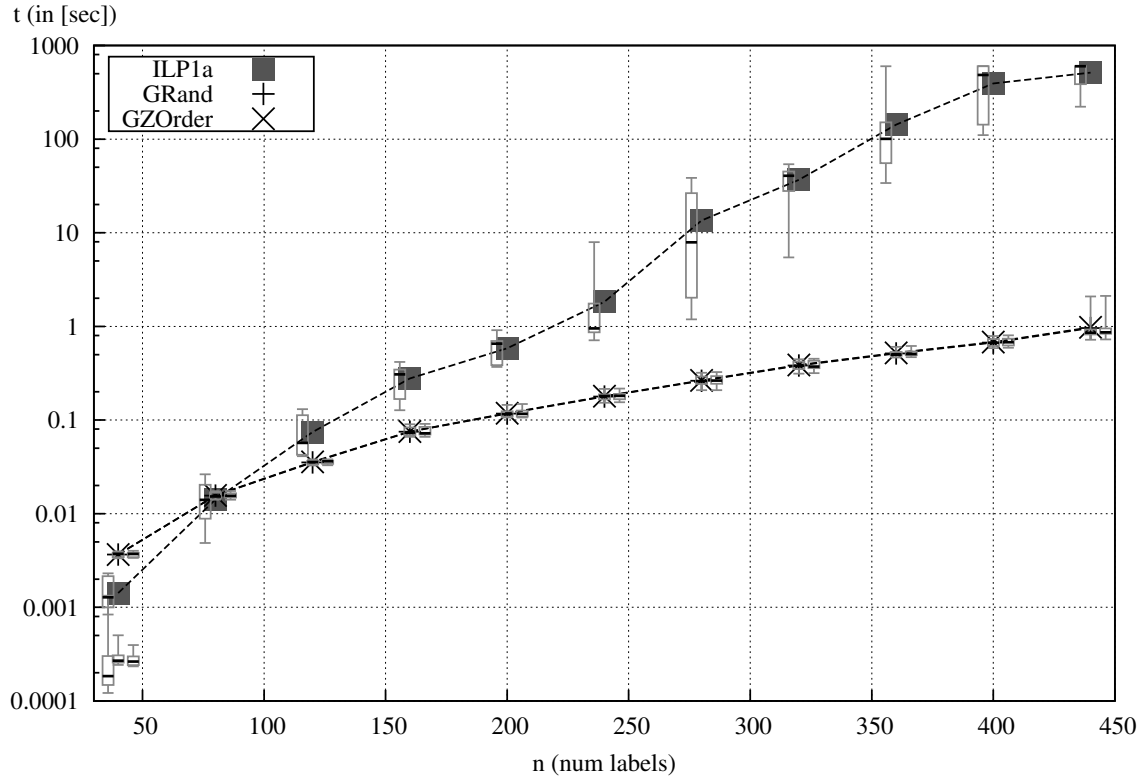


Figure 3.8: Greedy heuristics running time comparison, displayed on logarithmic scale. The boxplots are sorted within the groups from left to right: ILP1a, GRand, GZOrder.

Additionally to the objective value, we want to illuminate the solutions' quality from another perspective. As stated at the beginning of this chapter, we want to present the labels to the user in a most unobtrusive way. We aim to reduce the distraction due flickering of labels. This can be measured by the length of the labels' activity intervals, i.e. how long is a label shown to the user before vanishing. This allows us to estimate the volatility of the labelling solution.

In Figure 3.10, we show the average length of activity intervals of ILP1a, GRand and GZOrder. As the zooming ratio interval has a length of 10, this is the maximum value for the activity interval of a label. The activity intervals' length evidently reduces rapidly with an increasing number of labels. This is produces strongly observable flickering and leads to a bad dynamic behaviour of Labelling Model A.

To summarize the results, MILP and ILP2 lead for up to 50 data points within reasonable computation time to the optimal solution. Compared to this, the IPL heuristics are significantly faster but still very close to optimal solution.

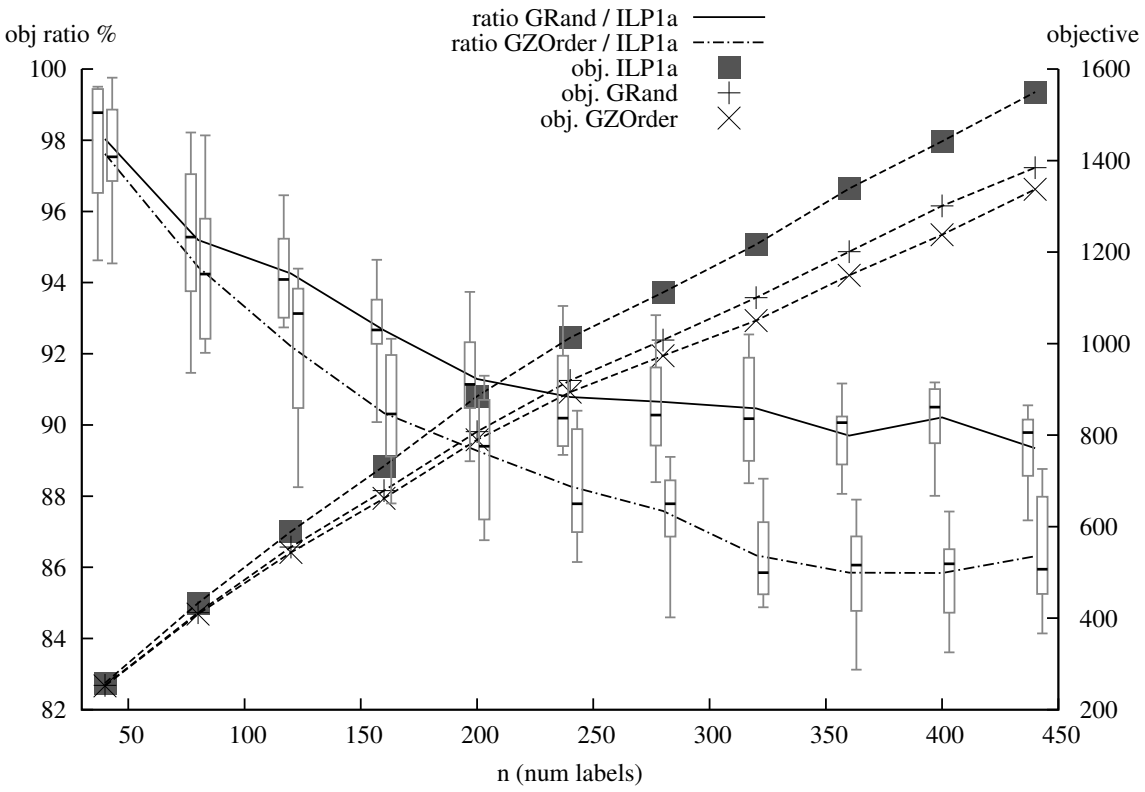


Figure 3.9: Greedy heuristics objective value comparison. The boxplots are sorted within the groups: left GRand, right GZOrder.

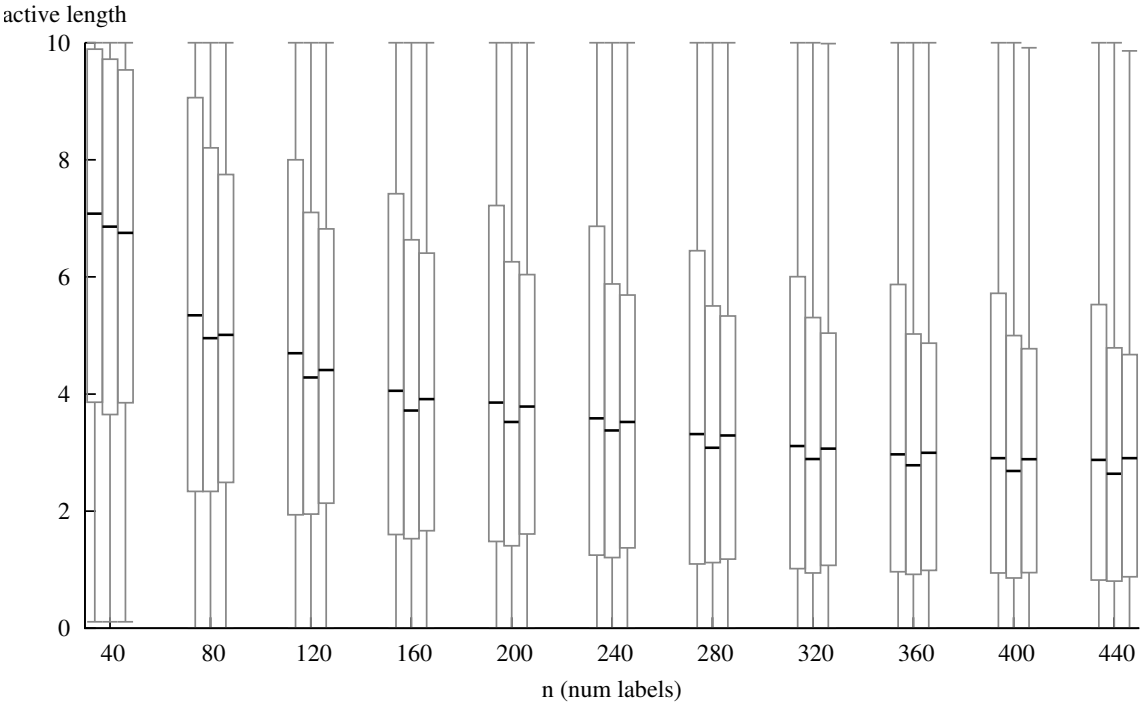


Figure 3.10: Heuristics activity interval length comparison. The boxplots are sorted within the groups from left to right: ILP1a, GRand, GZOrder.

The greedy heuristics are very fast and even for a large number of data points quite close to the optimal solution, which makes them suitable for real time applications and applications on mobile devices or for instances with more than 200 data points to be labelled.

The solution to Labelling Model A instances tend to short activity interval lengths for an increasing number of labels, which results in flickering and therefore a bad dynamic behaviour. To prevent this, we present a second, more restrictive labelling model in the next chapter.

4. Labelling Model B

In the previous chapter we allowed the labels to be switched on and off at two arbitrary zooming ratios, as long as the activity was one atomic interval. On one hand this allowed a big portion of the available screen and zooming ratio interval to be used. On the other hand, activity intervals could get very short. That produced a flickering impression as can be seen in the experiments in Section 3.6 although we particularly tried to avoid this.

In this chapter we will present another labelling model, which is more restrictive than the previously presented Labelling Model A, but in return reduces the amount of flickering. In addition, the introduced restrictions allow for further reasoning of theoretic results, and finally lead to an approximate algorithm to solve the ARO-3D on 2D screen problem.

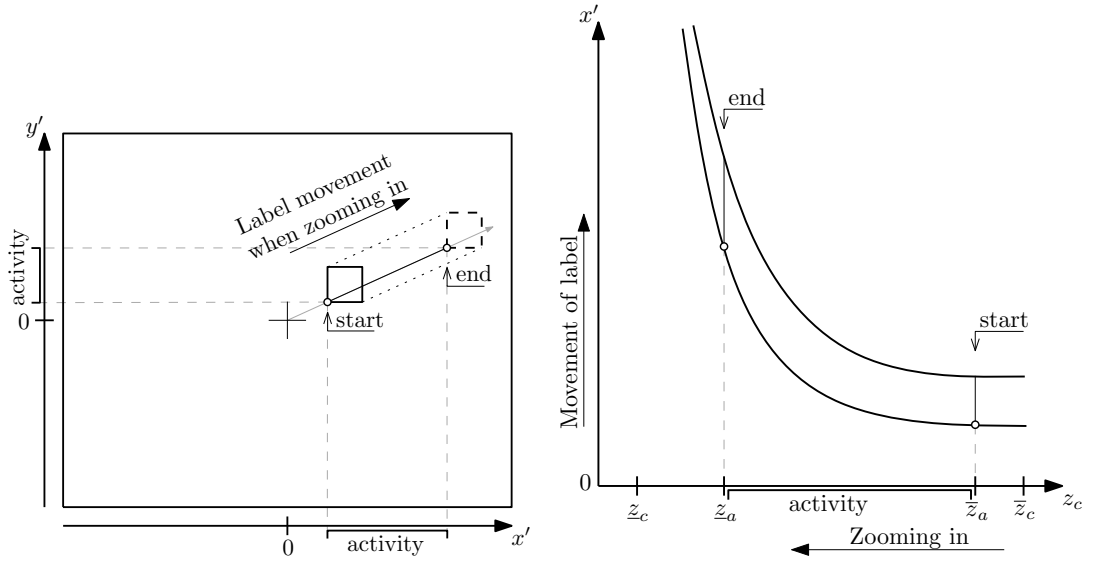
After initially describing this model, we will deduct a polynomial time algorithm that solves the ARO-3D on 1D screen problem in this labelling model optimally. The ARO-3D on 2D screen problem is NP-hard; we will present an approximate algorithm for this case.

4.1 The Model

For Labelling Model B, we again demand the activity functions of labels to be atomic intervals completely contained within a fixed zooming ratio interval $[\underline{z}_c, \overline{z}_c]$. That means Labelling Model B is basically the same as Labelling Model A. It differs only in when the activity of labels allowed to start and end.

To be able to distinguish between *start* and *end* of the activity interval, we consider the activity of a label as though zooming in continuously. That means the camera starts at the highest possible camera position $\overline{z}_c$ and continuously decreases its position z_c until it stops at the lowermost zooming ratio $\underline{z}_c$. When the camera reaches the upper bound $\overline{z}_a$ of label a's activity interval, this label is switched on in the viewport and stays active until the camera position reaches $\underline{z}_a$. The start of an activity interval $[\underline{z}_a, \overline{z}_a]$ then is the interval's upper bound $\overline{z}_a$, and hence the end is its lower bound $\underline{z}_a$, see Figure 4.1.

In Labelling Model B, only the activity interval's start may be chosen freely. Once activated, we demand the label to stay active when zooming in further. That means the activity interval's end is the smallest zooming ratio z_c, that is located within the zooming ratio interval and that is legal for the label, i.e. for which $z_c > z_a$ (see equation 2.6).



(a) Movement of label on 2D screen when zooming in

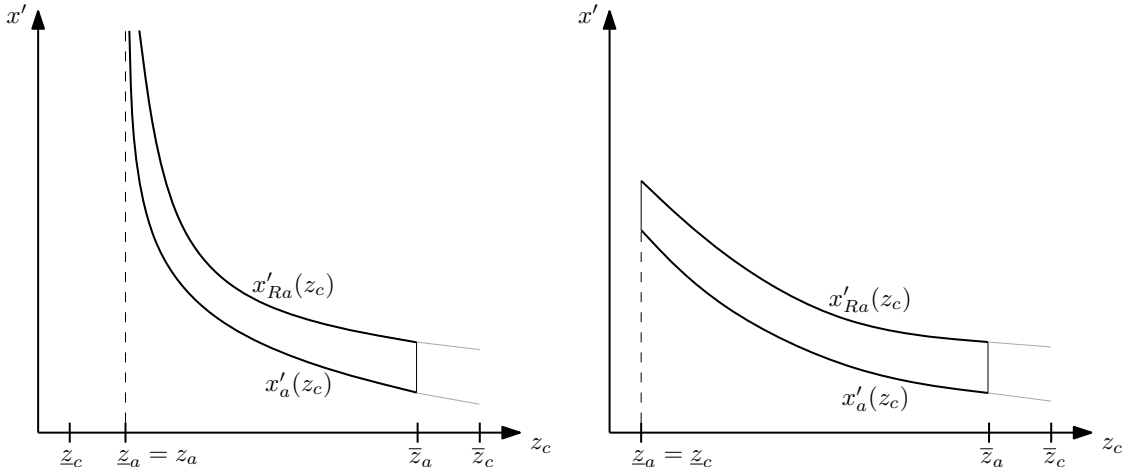
(b) x-Projection of label for varying zooming ratio z_c

Figure 4.1: Start and end of a label's activity interval when zooming in

For a data point a, given the starting point $\overline{z}_a$ of its activity interval and the zooming ratio interval $[\underline{z}_c, \overline{z}_c]$, we know that the activity interval's end is

$$\underline{z}_a = \max\{z_a, \underline{z}_c\} \tag{4.1}$$

and accordingly the activity interval of a label a itself is either $(z_a, \overline{z}_a]$ when z_a is located within the zooming ratio interval (see Figure 4.2(a)) or $[\underline{z}_c, \overline{z}_a]$ when $z_a < z_c$ (see Figure 4.2(b)).



(a) Activity interval ends at minimal legal z_c.

(b) $\underline{z}_a$ aligns with lower bound of zooming ratio interval.

Figure 4.2: The two possible endpoints of an activity interval starting at $\overline{z}_a$ in Labelling Model B.

This newly introduced restriction to the model does not affect the computational complexity of the arising ARO-3D labelling problem.

Theorem 2. ARO-3D *on a 2D screen in Labelling Model B is NP-hard.*

The reduction to prove this theorem is basically the same as the one in the proof of Theorem 1. The only difference for this proof is that labels stay active when zooming in, once they have been activated. But this is a basic assumption in [BNPW10] and does not introduce any additional constraints. The proof therefore is omitted here for the sake of brevity.

Despite its NP-hardness, Labelling Model B is still easier to solve than Labelling Model A: Since the activity intervals' endpoints are defined by equation 4.1, we only have to care for choosing the start points of the activity intervals. This reduces the number of possible different activity intervals per label from $O(n^4)$ (see deductions for ILP1) to $O(n^2)$.

Also finding the optimal start points of the activity intervals is easier, as the number of possible start points is smaller; we get a more strict version of Lemma 4. A label can only start at conflict events of conflicts, in which the very same label is involved. This again reduces the number of allowed activity intervals per label from $O(n^2)$ to $O(n)$.

Similar to Lemma 4, we consider the zooming ratio interval bounds to be conflict events too for the following Lemma.

Lemma 5. *We can find an optimal solution, in which the activity interval of each label starts at one of its conflict events.*

Proof. Given an optimal solution, we assume this solution contains a non-empty activity interval $[\underline{z}_a, \overline{z}_a]$ that starts between two conflict events of its own, which we call *start bounding events*. If there was no conflict with another label in between the two start bounding conflict events, the activity interval could easily be extended by increasing its start $\overline{z}_a$ slightly. The gain of this solution would rise without introducing an additional conflict (similar to the situation in Figure 3.3), which would be a contradiction to the solution's optimality.

So there has to be an active conflict with another label b which hinders label a from being extended. Since the conflicting label's activity interval hinders $\overline{z}_a$ from being increased, it must be right of $\overline{z}_a$: $\overline{z}_a \leq \underline{z}_b, \overline{z}_b$, but end between the two start bounding conflict events.

Due to equation 4.1, we know that $\underline{z}_b = z_b$, since otherwise, a's activity interval would be empty. Hence, the z-coordinate of label b's data point would be located within a conflict interval between a and b. This is a contradiction to Lemma 3. □

That means, we do not have to take into account indirect conflicts of other labels, as seen in the example in Figure 3.1, for possible activity interval starting points. Therefore the number of possible starting points per label is in $O(n)$.

Yet better, the number of relevant conflict events, at which the activity intervals can start, reduces to maximally one per pair of labels. The only relevant conflict event between two labels a and b is the lower bound of the last conflict interval $[\underline{z}^*_{ab}, \overline{z}^*_{ab}]$ that overlaps the zooming ratio interval $[\underline{z}_c, \overline{z}_c]$ (equation 4.3). If this lower bound itself is not contained within the zooming ratio interval, it is not possible to place both labels and we consider the minimal zooming ratio to be the pair's conflict event (equation 4.2). If there is no conflict at all between the two labels within the zooming ratio interval, we consider the maximal zooming ratio $\overline{z}_c$ to be the pair's only relevant conflict event per definition (equation 4.3, end).

$$\text{if } (\exists [\underline{z}^*_{ab}, \overline{z}^*_{ab}] \in C \mid \underline{z}^*_{ab} < \underline{z}_c < \overline{z}^*_{ab}) \text{ then } z^*_{ab} = \underline{z}_c \tag{4.2}$$

$$\text{else } z^*_{ab} = \min \left\{ \{ \underline{z}^*_{ab} \mid [\underline{z}^*_{ab}, \overline{z}^*_{ab}] \in C : \underline{z}_c < \underline{z}^*_{ab} \} \cup \{ \overline{z}_c \} \right\} \tag{4.3}$$

Lemma 6. *Given two labels and all their common conflict events, in the optimal solution any of the two labels' activity intervals starts either at the last of their common conflict*

events within the zooming ratio interval (the one with minimum z) or at none of them at all.

Proof. Assume a valid solution in which both labels start at conflict events that are not the last within the zooming ratio interval. There is at least one other conflict event to come when zooming further in. This conflict event (as every other conflict event) is adjacent to a conflict between the two labels (see *canonical set* Section 2.5). Since the two labels will continue existing in Labelling Model B, they will have an additional conflict. That is a contradiction to the solution's validity.

Assume one of the labels start at a conflict event previous to the last. Due to the general position, at this previous conflict event no other conflict starts or ends. That means no other conflict relation changes. Since the other label is not active at this point (excluded above), it is safe to move the activity interval by an $\varepsilon > 0$ without introducing an additional conflict. But especially extending the label's activity interval (i.e. increasing its upper bound) would increase the solution's gain. This is a contradiction to the optimality of the solution. □

That means for each label we have only n conflict events that are relevant for the decision, where this label should start. The ARO-3D for Labelling Model B now reduces to the question, which label should be activated at which of it's conflict events so that we produce a conflict free, maximal solution.

In the following section, we will take a look at a solution to this problem on a 1D screen. The ARO-3D for the 1D screen is not NP-hard and we can find an optimal solution in polynomial time. Afterwards we'll derive an approximation algorithm for the 2D screen problem from this solution.

4.2 ARO-3D on 1D Screen Optimal Solution

In ARO-3D on a 1D screen, a set of 2D data points $S \subset \mathbb{R}^2$, having x and z coordinates, are projected onto a 1D line. Labels attached to those data points then are intervals on that line. We want to avoid the intersection between two labels, which is equivalent to overlapping of their intervals on the 1D screen.

Hence, when placing a label on the 1D screen, this splits the screen into a left and a right side. Since in Labelling Model B the label will stay active when zooming in further, this subdivision into left and right side will persist.

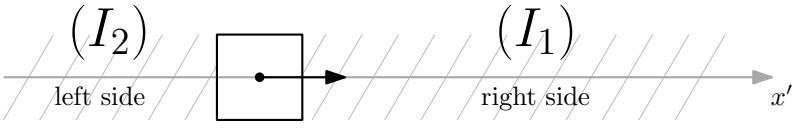


Figure 4.3: Splitting 1D screen into left and right side by an active label

From the start of the label's activity, no other label can pass this label (from left to right) without intersecting it. But since the intersection of active labels is not allowed in Labelling Model B, no other label can be active on the left *and* right side.

Recall that we demand in Labelling Model B that a label stays active once it has been activated. Hence it is not allowed to activate a label, if there is a conflict with an already active label to come when zoomed in further. In the example in Figure 4.4, label a is already activated and splits the screen in a left side (I_2) and a right side (I_1). Label b is only allowed to be activated in I_2, while label d is only allowed to be activated in I_1; they

cannot intersect. The placement of label a therefore splits the problem in two independent subinstances.

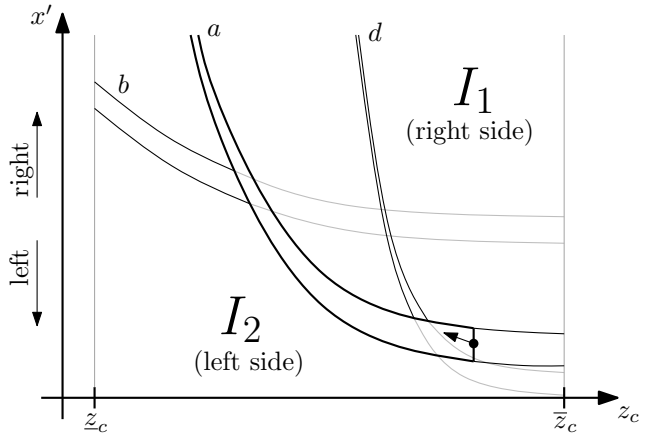


Figure 4.4: Example with label a separating labels b and d and therefore splitting problem in two independent instances.

To determine, which label is in the left or right subinstance of an active label, we refer to Lemma 3. Therefore a label b is in the left subinstance of an active label a, only if $z_b < z_a$. Otherwise b is in the right subinstance. The case $z_b = z_a$ cannot occur as we assume general position of the data points.

Considering the example in Figure 4.5, activating a label is not enough to split the instance in two separate subinstances. Here despite of the active label a splitting a part of the instance into a left and a right side, the trajectories of labels b and d still intersect.

But if we restricted the subinstances to the part of the screen which is right to the label when activated, i.e. to $x' > x'_a(\overline{z}_a)$ (see grey dashed line in Figure 4.5), we could assure independency of the two subinstances. Due to the strict monotonic behaviour of the labels' trajectories there is no zooming ratio for which the labels b and d could intersect.

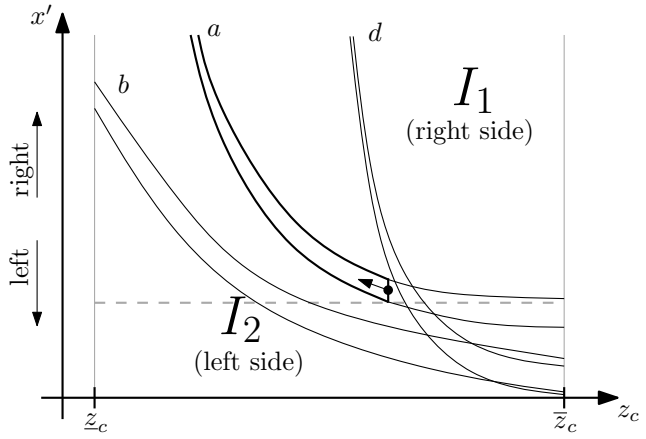


Figure 4.5: Example with label a starting to "late" to separate labels b and d.

If we actually did leave out that left part of the screen (and therefore of subinstances), we would certainly take a big loss compared to the optimal solution. Hence we have to find a way to assure that there is nothing left to the restriction line which we would lose compared to the optimal solution

Due to their general position the labels' activity intervals start at different x' positions in the optimal solution. So they can be sorted unambiguously in ascending order. If we knew

this order, we could easily construct optimal solution by activating the labels in this order at the leftmost legal position.

This observation gives rise to a dynamic program formulation: we simply guess that order. Given the label with lowest x', placing this label divides instance in two truly independent subinstances. For each of those subinstances, choosing lowest label to be placed next, splits the subinstance in independent sub-subinstances and so on.

In general, a subinstance I of the dynamic program is given by a left bounding label a, a right bounding label b and the lower bound of the (sub-)instance. The lower bound is not given by a continuous x'-coordinate, but can be identified by a conflict event z^*_{ha} or z^*_{hb} of either the left, or the right boundary label, with some other label h, so that it is rightmost activity interval starting point of the two bounding labels, see Figure 4.6. This yields the notation $I_{a,b,z^*_{hb}}$ for a subinstance.

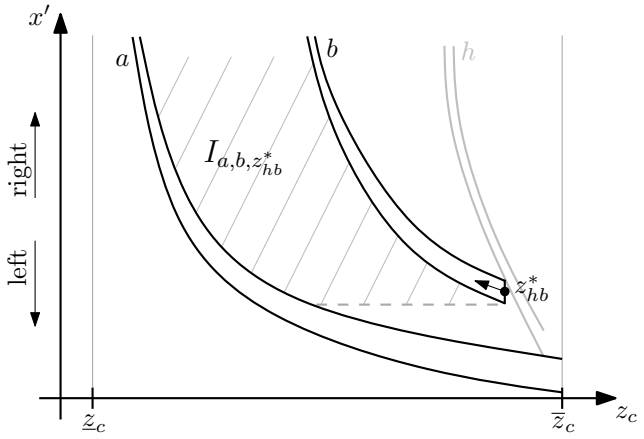


Figure 4.6: Example of a subinstance induced by labels a and b and conflict event z^*_{hb}.

The gain value of an instance $I_{a,b,z^*_{hb}}$ with respect to the ARO-3D gain function is the activity length $(\overline{z}_j - \underline{z}_j)$ of the next label j splitting this instance in two *plus* the values of the two subinstances induced by the bounding labels of this instance and the splitting label j. The table of the dynamic program contains one entry for each subinstance.

$$T[a,b,z^*_{hb}] = \max_{j=1..n, i\in\{a,b\}} \left\{ (\max\{\underline{z}_c, z_j\} - z^*_{ij}) + T[a,j,z^*_{ij}] + T[b,j,z^*_{ij}] \;\middle|\; z^*_{ij} \in E_p \right\} \tag{4.4}$$

with E_p determining the set of all conflict events in the instance. We will soon have a closer look on that one.

The first two indices for a table entry are the left and right bounding labels, each of which can take n possible values. The third parameter, the conflict event determining the instances lower bound, takes $O(n^2)$ different values when implemented naively, since there are $O(n^2)$ possible conflict events (see Lemma 1). But we certainly know, that this conflict event is related either to the left, or to the right bounding label. Since the number of conflict events of a certain label is in $O(n)$, the number of possible lower bounds reduces to $2 \cdot O(n) = O(n)$.

Moreover, we can encode the information, with which of the two bounding labels this conflict event is associated, into the order of the first two indices, i.e. having the conflict event always associated with the second of the two labels in the input argument list, which leads to table entries $T[a,b,z^*_{hb}]$ and $T[b,a,z^*_{ha}]$ for bounding labels a and b.

As we have shown in Lemma 6, between two labels there is only one conflict event relevant for the optimal solution in Labelling Model B. Hence referring to a conflict event of a label

b by the index of the conflicting label h suffices completely to unambiguously identify the conflict event z^*_{hb}.

Therefore a table entry can be identified by the indices of the two bounding labels and the index of a third, conflicting bounding label. In the following it will be enough to write table entries as $T[a, b, h]$. The total complexity of the table is in $O(n^3)$.

Additionally it is necessary to introduce a conflict event for each label for the upper zooming ratio interval bound $\overline{z}_c$. That allows labels to be activated at the very beginning of zooming in. In the table we will denote them by $T[a, b, \perp]$ for a label b with its activity interval beginning at $\overline{z}_b = \overline{z}_c$.

And we need additional dummy labels for the zooming ratio boundaries $\underline{z}_c$ and $\overline{z_c}$, which we will denote by $\top$ and $\perp$. So, all in all we are interested in the value of the table entry $T[\perp, \top, \perp]$, which we will calculate by recursively calculating the whole table.

The last question left to answer is which possible splitting labels (conflict events) are allowed to split an instance $I_{a,b,z^*_{hb}}$ in two subinstances. We already stated that the first condition to be satisfied is, that the label splitting the instance has to start at a conflict with either the left, or the right bounding label. That means we are maximizing in equation 4.4 over $O(n)$ possible splitting labels.

The second condition to be satisfied is that the start of the splitting label has to be located within the region defined by the instance $I_{a,b,z^*_{hb}}$. That is, how is the set E_p characterized which determines over which elements the table entry in equation 4.4 is maximized. As stated in Lemma 6 there exists only one relevant conflict event per pair of labels. And the starting point of splitting label must lie within the instance's region, otherwise it would either intersect with one of the boundary labels or its starting point would lie below the lower bound and label b was not the lowest label in the optimal solution.

We will use a mathematical "shortcut" here. The projection trajectory of a label a is defined as

$$x'_a(z) = x_a \cdot d \cdot \frac{1}{z - z_a} \qquad , z > z_a \tag{4.5}$$

which is invertible for $z > z_a$ and we can define the reverse function

$$z'_a(x) := x_a \cdot d \cdot \frac{1}{x} + z_a \qquad , x \neq 0 \tag{4.6}$$

so that

$$x'_a(z) = x \Leftrightarrow z'_a(x) = z \ . \tag{4.7}$$

In equation 4.4 we used the set E_p of legal conflict events z^*_{ij} within the region defined by a, b and z^*_{hb} (i.e. the region of an instance $I_{a,b,z^*_{hb}}$). Such legal conflict events z^*_{ij} have to be enclosed by the two bounding label trajectories $x'_a(z)$ and $x'_b(z)$, which yields

$$z'_a(x'_j(z^*_{ij})) < z^*_{ij} \leq z'_b(x'_j(z^*_{ij})) \qquad \text{or} \qquad z'_b(x'_j(z^*_{ij})) < z^*_{ij} \leq z'_a(x'_j(z^*_{ij})) \tag{4.8}$$

depending on the order of the bounding labels a and b. Additionally, the legal conflict events have to satisfy the instance's lower bound, which is given by z^*_{hb}:

$$x'_j(z^*_{ij}) > x'_b(z^*_{hb}) \ . \tag{4.9}$$

Putting those constraints together, E_p is then given by

$$\text{if } (z'_a(x'_b(z^*_{hb})) < z^*_{hb}) \tag{4.10}$$

$$\text{then } z^*_{ij} \in E_p \Leftrightarrow z'_a(x'_j(z^*_{ij})) < z^*_{ij} \leq z'_b(x'_j(z^*_{ij})) \wedge x'_j(z^*_{ij}) > x'_b(z^*_{hb}) \tag{4.11}$$

$$\text{else } z^*_{ij} \in E_p \Leftrightarrow z'_b(x'_j(z^*_{ij})) < z^*_{ij} \leq z'_a(x'_j(z^*_{ij})) \wedge x'_j(z^*_{ij}) > x'_b(z^*_{hb}) \ . \tag{4.12}$$

Hence when maximizing over all conflict events over all splitting labels starting on the left and right bounding label, we can verify in constant time, that the corresponding label is in E_p. Since having to maximize over $O(n)$ table entries to calculate table entry $T[a, b, h]$, calculating this table entry is possible in $O(n)$ time. As stated above, we have a total of $O(n^3)$ table entries hence calculate all entries takes $O(n^4)$ time. The optimal value of the initial instance is then in entry $T[\bot, \top, \bot]$.

Theorem 3. *The above constructed dynamic program computes the optimal solution of ARO-3D on a 1D screen in Labelling Model B in $O(n^4)$ time and $O(n^3)$ space.*

From this result, we will step by step construct an approximate algorithm for the ARO-3D on 2D screen problem throughout the following two sections.

4.3 The Radial Stabbing Tree

Before we are going to extend the optimal solution of ARO-3D on 1D screen to an approximate algorithm for ARO-3D on 2D screen, we will at first present a technique which allows us to decouple label-placement decisions for non-intersecting labels on the 2D screen.

The basic idea behind the approach presented in the following is that two labels can only have a conflict if they are directly neighboured. Hence when deciding that one of the labels is placed and the other one is not, this is a very local decision. Labels being located very far from each other on the screen actually do not have a direct conflict and therefore do not have a direct influence on their mutual placement. The NP-hardness proof relies on the property that label conflicts can be chained, so that very distant labels could have an influence on each others placement without actually having a direct conflict.

The second observation relevant for this approach is that when placing a label this label occupies a certain space on the screen. So, if many labels have to be placed in a very small area of the screen, the number of labels that actually can be placed is bound rather by the available area of the screen than by the number and constellation of labels.

To incorporate these two ideas into a solution, we decouple placement decisions for labels located very far from each other on one hand; and on the other hand we aim for bounding the area of the screen in which labels are actually placed so that the number of labels is not relevant for an approximation factor, but rather the size of the area of the screen containing the labels.

For this purpose, in this section we describe a technique to divide the 2D screen into several independent sectors. This results in a tree-like structure, which we call *Radial Stabbing Tree*.

The sectors of the radial stabbing tree implicitly define simplified subinstances of the actual ARO-3D on 2D screen labelling problem which we assume can be solved optimally. To make a first step towards the approximation algorithm presented in the next section, we will at the end of this section bound the loss due to the subdivision of the screen into sectors.

Radially Aligned Labels

To ease the theoretical analysis in the following, we slightly change the assumption on the label shapes. On one hand, we assume all labels to be unit squares. Secondly, we drop the constraint that all labels are axis aligned. Instead, they are aligned with the direction into which they move on the screen. This causes all labels to be rotated. Third, and finally, the labels' anchor points are assumed to be located at the very center of the label. See Figure 4.7 for an exemplary sketch of this new label setup.

The width of the label, perpendicular to the label's direction of movement, is called the label's diameter d_L. It is constant and for all labels the same. Vice versa we call the label's extent in its direction of movement the label's height, which is obviously equal to the label's width. In this manner we additionally introduce the lower and upper boundary of the label, which denote the edge of the label nearer to the center of the screen and edge further away respectively (cp. Figure 4.7).

Note that the results from Section 4.1 and Section 4.2 can easily be extended for the new label shapes. The theoretic results we deduct in the following can be extended for other label shapes with an in-depth geometrical analysis, which we will omit in this work for the sake of brevity.

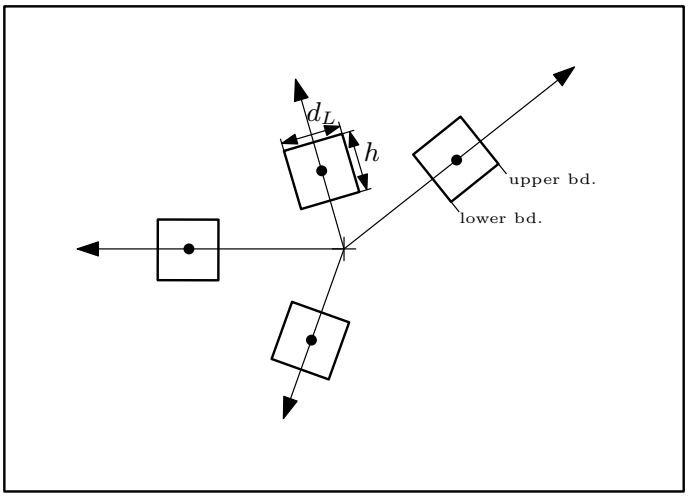


Figure 4.7: The square labels are rotated to align with their on-screen trajectory.

Polar Conflicts

A second change to facilitate the analysis in the following concerns the notion of the labels' coordinates. We no more view the position of a label a as projected (x'_a, y'_a)-coordinates but rather as projected distance d'_a from the center of the screen and a rotation ϕ'_a relative to the x'-axis (see Figure 4.8). That means we now use a polar coordinate system instead of the Cartesian system.

Taking into account the newly defined label shapes, the label's height now defines an interval on the projected distance coordinate d'_a, bounded by the label's lower and upper boundary. In this new notation two labels have a conflict if their height-intervals overlap and their rotation is "almost" the same, depending on the labels' distances from the center of the screen and diameter. Since this is not a functional novelty, but rather a change of the mathematical viewpoint on the basic labelling model introduced in Chapter 2, we will at this point omit the geometric details for the sake of brevity.

Levels and Line Segments

At first the screen is divided into different *levels* by means of concentric circles with their midpoints being the center of the screen. The innermost circle has index $k = 0$ and its area defines level zero. The ring between the innermost and the second circle (indexed with $k = 1$) defines level one. We proceed in this manner, so that the ring between circles k and $k + 1$ referred to as level $k + 1$, and this level is bounded by those two circles.

Additionally we place radial line segments in each level, which we will use later on to define independent sectors. The line segments reach from the inner circle bounding that level to

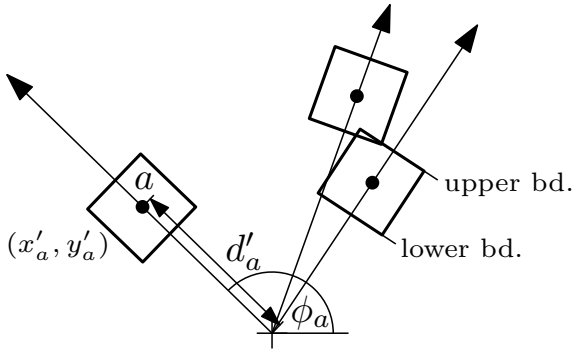


Figure 4.8: Polar coordinate system for label trajectories.

the outer circle and are perpendicular to the circumferences. In Level k we place 2^{k+1} line segments equidistantly, except for level 0, where no line segments are placed. See Figure 4.9 for a sketch of this setup.

That means when transitioning from level k to level $k+1$, the number of line segments is doubled. The idea behind this construction is, that the area between two line segments within the same level defines a sector (more or less), which is split up in two sectors in the next level. This construction leads to the previously mentioned binary tree structure: Each sector at a level k is followed by two child sectors at level $k+1$, which in turn are followed by two child sectors at level $k+2$, and so on.

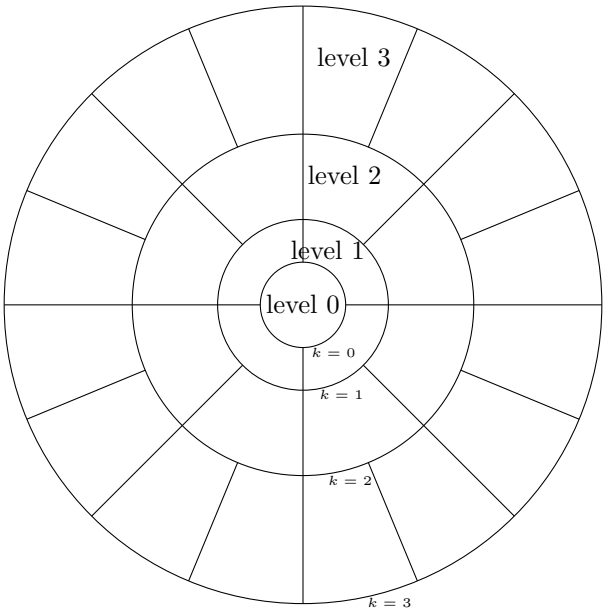


Figure 4.9: Basic construction of the Radial Stabbing Tree by means of levels and line segments.

For this construction to work properly later in this chapter, the circles have to be placed at certain distances from the center. The innermost circle has radius

$$r_0 = \frac{d_L}{\sqrt{2}} \tag{4.13}$$

with d_L being the labels' diameter (which we assume to be equal for all labels). All outer circles k will be placed at radiuses

$$r_k = \frac{d_L}{2 \cdot \sin(\gamma_k/4)} \text{ with } \gamma_k = \frac{\pi}{2^k} \ . \tag{4.14}$$

This construction aims for all line segments of the same level to have a minimum distance of d_L. Since the line segments are perpendicular to the circles, the point of minimal distance of two neighbouring line segments is at their ends pointing towards the center of the screen. Hence we calculate their distance by means of the inner circle's chords between the points of the circle touching the line segments' endpoints (cf. Figure 4.10).

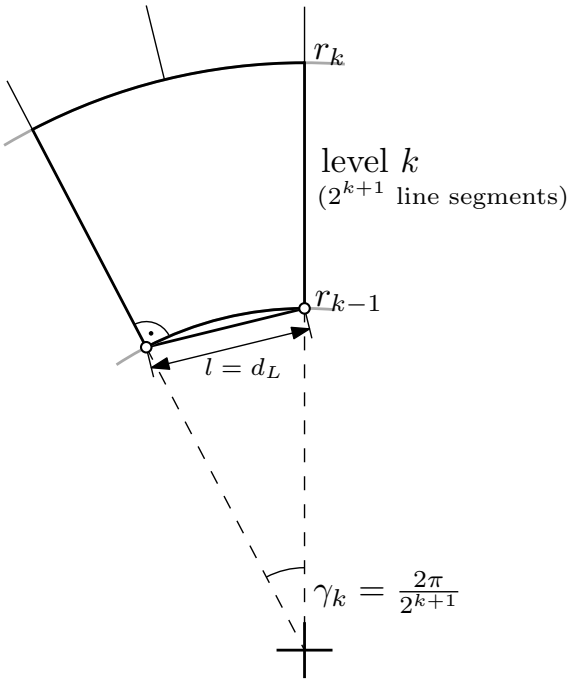


Figure 4.10: Construction to determine minimum distance between line segments.

We now show that choosing the radius r_{k-1} of circle $k-1$ as described above results in the line segments at level k having minimum distance d_L. The 2^{k+1} line segments in level k are placed equidistantly all around the inner circle $k-1$, which causes the line segments to be aligned at angles of

$$i \cdot (2 \cdot \pi)/(2^{k+1}) + \widetilde{\gamma} \qquad , i = 1, \ldots, 2^{k+1} \tag{4.15}$$

with $\widetilde{\gamma}$ being a circular offset. That means the angle between two neighbouring line segments at level k is

$$(2 \cdot \pi)/(2^{k+1}) = \frac{\pi}{2^k} = \gamma_k \tag{4.16}$$

which we will use to calculate the length of the chord between points at those angles on level k's inner circle $k-1$:

$$l = 2 \cdot r \cdot \sin(\gamma/2) \tag{4.17}$$

with r being the circles radius and γ being the angle between the two points defining the chord. That means using the definitions from equation 4.14 (and the fact that $\gamma_k = \gamma_{k-1}/2$) we obtain the length of the chord as

$$l = 2 \cdot r_{k-1} \cdot \sin(\gamma_k/2) = 2 \cdot \frac{d_L}{2 \cdot \sin(\gamma_{k-1}/4)} \cdot \sin(\gamma_k/2) = d_L \tag{4.18}$$

which is exactly what we were aiming for.

Stabbing Label Groups

When now embedding the labels' on-screen trajectories to this construction we easily see that, when zooming in, the labels move from innermost level at the center of the screen towards outer levels with increasing indices. Since the minimum distance between two line segments is d_L, the labels' diameter, we also know that within one level the labels' trajectories do intersect with only one of the level's line segments. Only in the rare case, that the label passes exactly in the middle between two line segments, its boundary touches the left and right line segment at the same time as depicted in Figure 4.11. In this case we "manually' reduce the number of intersections to one by ignoring the intersection between the label and the line segment in clockwise direction. As the label cannot pass the level's inner bounding circumference without intersecting with at least one line segment (since the distance between two line segments is d_L at this point), each label trajectory therefore has exactly one intersection with a line segment per level.

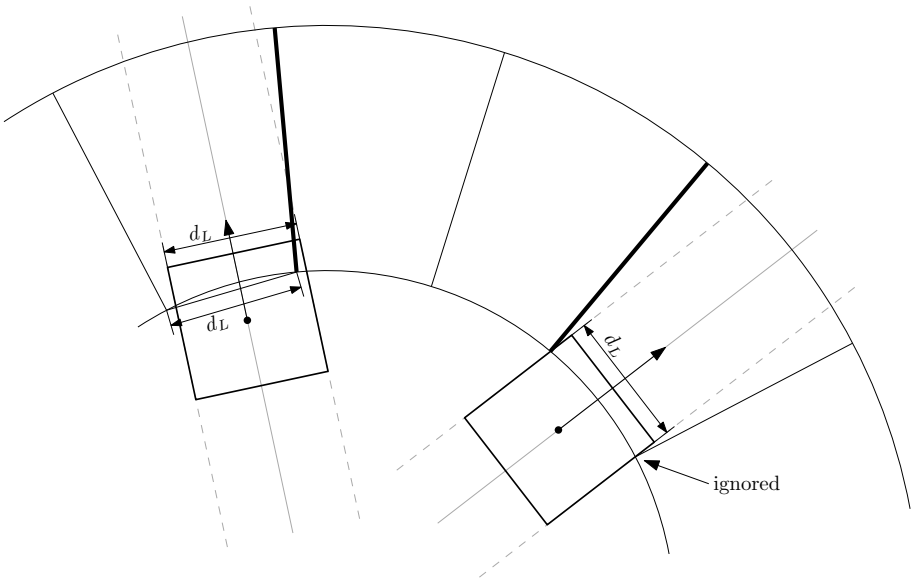


Figure 4.11: Each label trajectory intersects with exactly one line segment per level

We divide the line segments of one level into two groups so that, radially spoken, every second line segment is in the same group. Since the number of line segments within one level is even, no two segments of the same group are neighboured. This similarly divides the labels into two groups, as each label's trajectory overlaps with exactly one line segment in this level: all labels whose trajectories overlap with line segments of the same group form one label-group.

This division of the labels into two disjoint groups yields two interesting properties: Label trajectories that are part of the optimal solution are split into two groups too. Hence one of the two label groups contains at least half of the labels of the optimal solution. That means, if we could calculate optimal solutions on both groups separately, the better solution of the two would be at least half as good as the optimal solution.

The second property concerns the independence of labels intersecting with different line segments. Since all label trajectories intersect with exactly one line, a label trajectory cannot intersect with two lines of different groups. And because two line segments of the same group are always separated by a line segment of the other group, the line segment in between works as kind of a separator between the label trajectories intersecting with those two line segments of the same group. No label can pass the line segment in the middle and intersect with a label on the other side of this separator. That means due to this separating

line segment, we have two independent sub-problems to solve when placing labels on this level. We call the areas around the active line segments, separated by the line segments of the other group in between, *sectors*. The labels of the other group are ignored when searching a solution for this group. Therefore, when searching for an optimal solution for this group on this level it will suffice to solve conflicts in the sectors independently.

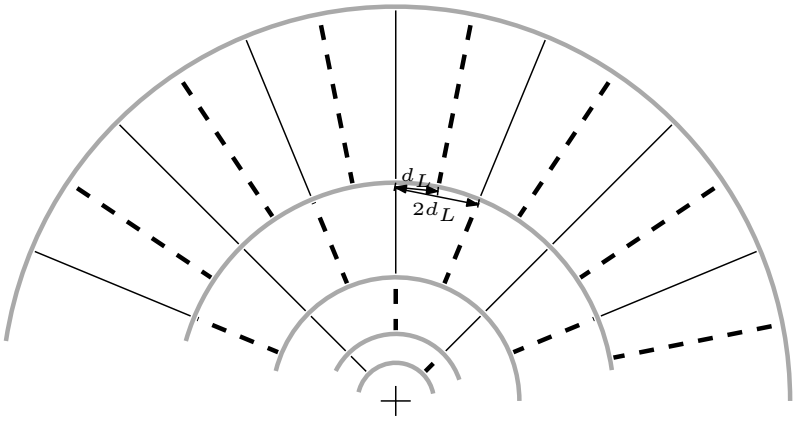


Figure 4.12: At each level, neighbouring sectors of the same group (heavy, dashed lines) are separated by the line segments of the other group (thin, continuous lines).

We now reconsider the idea from above that we have two sets of line segments, which yield two disjoint groups of labels and therefore two disjoint problem instances to solve and choose the better one. Having introduced the notion of sectors, choosing between the two instances is equal to choosing between two different subdivisions of the level into sectors divided by the set of non-chosen line segments. Each sector for itself is independent from its sibling sectors of the same group on the same level. Thus, when calculating the solution for one group in this level, we simply solve each sector independently and accumulate those solutions for the group and sum up their individual gain values for the gain value of the group.

The Tree

Up to now we only regarded one level separately and assumed we were interested in an optimal solution for this level only. Under the assumption we could solve the independent sector-subinstances optimally, this allowed us to choose between two solutions to obtain an approximation of the actual optimal solution. But the overall situation is not that simple as we want to calculate a solution for the whole screen and label trajectories being independent within one level might not be independent at other levels, especially at lower levels. Closer to the center of the screen, the labels' trajectories, previously separated by line segments, approach each other and can introduce additional conflicts. To cope with that situation and resolve conflicts on the different levels simultaneously, we finally introduce the concept of the *Radial Stabbing Tree*.

Each sector is equivalent to a node in the radial stabbing tree. The root node of the tree is the innermost sector at level zero of the screens subdivision. Since at level zero there are no line segments to split it up into sectors, there is only this one, unambiguous sector. Level one has four line segments which result in two different subdivision each with two equal sized sectors. This yields two versions of the radial stabbing tree, in each of which the two sectors at level one are child nodes of the root node in the tree.

When proceeding to the next higher level, the number of line segments (and therefore the number of sectors) doubles, as mentioned previously. For the moment we will assume that

the splitting lines of the next level are aligned with the previous level's line segments. That means each sector is split in two at the next level and thus, each sector node has two child sector nodes in the stabbing tree. That means, given the k levels on the screen (i.e. the outermost level is level $k-1$), the overall number of sectors in the radial stabbing tree is $2^k - 1$.

Approximation Factor

Despite this definition of the radial stabbing tree from its root to the leaves, the actual stabbing tree (and the solution) is constructed from the outermost level, i.e. the leaf-level of the radial stabbing tree, to the root node.

Starting at the outermost level, we have the choice between two disjoint configurations, one of them containing at least half of the labels of the optimal solution. We assume we could guess which of the two subdivisions contains the greater part of the optimal solution and discard the other labels. Thus we obtain a set $S_k \subset S$ with $g(S_k) \geq \frac{1}{2} g(S)$, where $g(S)$ denotes the gain value of the optimal solution for the corresponding set of labels.

Now we proceed with the set S_k of remaining labels to the next lower level. Here again, two subdivisions of this level yield a decomposition of the remaining set into two disjoint subsets. One of them contains at least half of the labels of the optimal solution from the remaining set: $g(S_{k-1}) \geq \frac{1}{2} g(S_k)$.

In this manner we proceed for all levels until we reach the innermost level. At each level, we keep at least half of the remainder of the optimal solution, the rest is discarded. That means the gain of the optimal solution is (in worst case) halved at each level, leading to a total gain of the remaining of $g(S_0) \geq \frac{1}{2}^k g(S)$.

This approach leaves the choice between two different subsets at each level. Guessing which subset to take (as assumed) obviously does not lead to a deterministic algorithm and hence not necessarily to the right solution. Locally choosing the subset that has a better optimal solution at this very level does not necessarily solve this problem either. The overall optimal solution might not be locally optimal at this level due to interdependencies between levels.

Instead we take kind of a brute force approach. We try all combinations of choices between the two subsets of each level. When having k levels this results in k choices each between two alternatives, which makes a total of 2^k different subdivisions, i.e. different subsets on which to solve the ARO-3D problem. We then choose among the 2^k subset-solutions the one with the best gain value, which is a $\frac{1}{2}^k$ approximation of the optimal solution following the reasoning above.

Constant Screen Size

For the stabbing tree approximation approach to work, we assumed the existence of an outermost level k. In real world applications, the size of the screen presented to the user is given by width w_s and height h_s and presumably constant. Similarly we assume the labels' diameter to be given by a constant and not to be changed. We now choose k so that its radius is

$$r_k \geq \sqrt{(\frac{w_s}{2})^2 + (\frac{h_s}{2})^2} \,, \tag{4.19}$$

which means that the whole screen is covered by levels zero to k, see Figure 4.13.

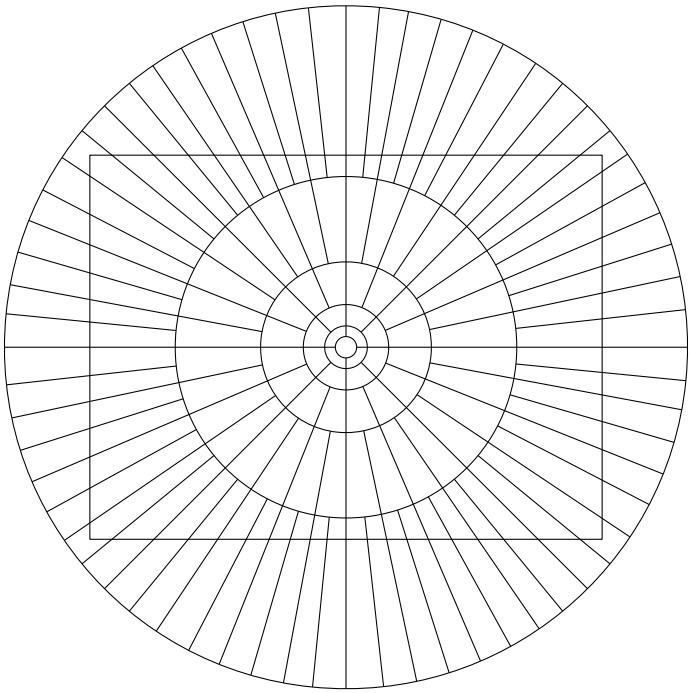


Figure 4.13: The Radial Stabbing Tree covers the screen of constant size. In this example with a reasonable screen to label size ratio we need $k = 6$ levels.

4.4 ARO-3D on 2D Screen Approximation Algorithm

In this section, we use the previously presented Radial Stabbing Tree to construct an approximate solution for the ARO-3D on 2D screen problem. We assumed that we could solve the resulting ARO-3D problem for the labels in the stabbing tree optimally and showed that this would result in a $1/2^k$-approximation of the ARO-3D on 2D screen problem. Instead of an optimal solution, we present a 1/3-approximation based on the ARO-3D on 1D screen optimal algorithm presented in Section 4.2. In total, this leads to a $1/(3 \cdot 2^k)$-approximation for the ARO-3D on 2D screen problem.

To be able to make use of the ARO-3D on 1D screen optimal algorithm, we at first have to reduce the 2D sectors, which are subdividing the 2D screen, to 1D screens.

Reducing Sectors to 1D Screens

The characteristic property of the ARO-3D on 1D screen problem, which we need for the algorithm to work, is that two labels having the same distance from the center of the screen always overlap. In the 2D sectors however, labels can be active in parallel without overlapping. But due to the specific geometry of the sectors, no more than three labels with the same distance can be active independently without introducing additional conflicts. This is due to the fact that the distance between the left and the right separating line is less than four times the labels' diameter and labels being allowed to be active in this sector must not overlap with the sectors separators. This observation leads to the following construction.

We place three stabbing lines in a sector, which are aligned with the labels' direction of movement (see Figure 4.14, dashed lines) so that each label is stabbed by at least one stabbing line for the whole sector. Obviously, all labels stabbed by the same line at the same distance have a conflict with each other at this distance, since they overlap simultaneously with the stabbing line. That means all labels stabbed by the same line form an ARO-3D

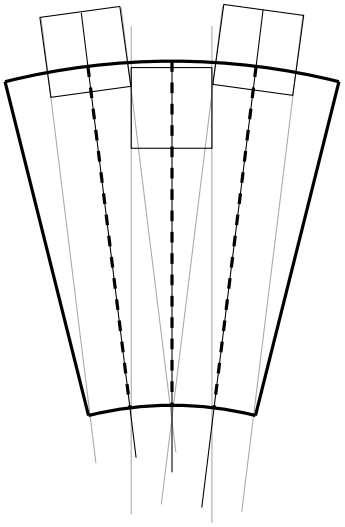


Figure 4.14: Within one sector, only three labels can be active while having the same distance from the center of the screen and not being stabbed by the sector's borders.

on 1D screen instance, since they have a conflict when occupying the same area of the 1D screen, which in this case is the corresponding stabbing line.

Since each label in this sector is stabbed by at least one of three stabbing lines *for the whole sector*, these lines induce three (not necessarily disjoint) groups, each of which can be seen as a ARO-3D on 1D screen instance. Those three groups cover the whole set of labels in this sector. When considering the part of the optimal solution in this sector, one of the groups contains at least one third of the optimal solution.

Hence when solving this whole (sector) instance optimally under the assumption it was a 1D screen instance (i.e. labels with same distance d' from the center of the screen have a conflict) we obtain a solution with at least a one third of the optimal solution's gain on the original 2D screen sector.

Now the time has come to adapt the ARO-3D on 1D screen optimal solution from Section 4.2 to the newly arisen ARO-3D problem on the 1D screens tree.

1D Screen Tree

Having reduced the sectors of the Radial Stabbing Tree to 1D screens, the whole 2D screen is now presented by a tree of 1D screen segments (see Figure 4.15). When zooming in, starting from a very high zooming ratio, a label starts at the innermost sector and then proceeds to one of the sector's child sectors. It passes through the whole tree always transitioning from a sector to one of its child sectors, until it reaches a leaf node and therefore leaves the screen. Hence a label trajectory defines a complete branch of the stabbing tree and therefore a series of 1D screens.

On the other hand this means, that on the root sector all labels are located on the same 1D screen and then are split up in two groups when transitioning to the next higher level. These two groups are completely independent of each other on this level. They are again split up into two groups when transitioning at the upper bound of this level to two sectors of the next level. This situation is roughly sketched in Figure 4.16 and Figure 4.17.

In the simple ARO-3D on 1D screen optimal algorithm, the placement of a label splits the 1D screen in two independent subinstances, which we used to construct a dynamic

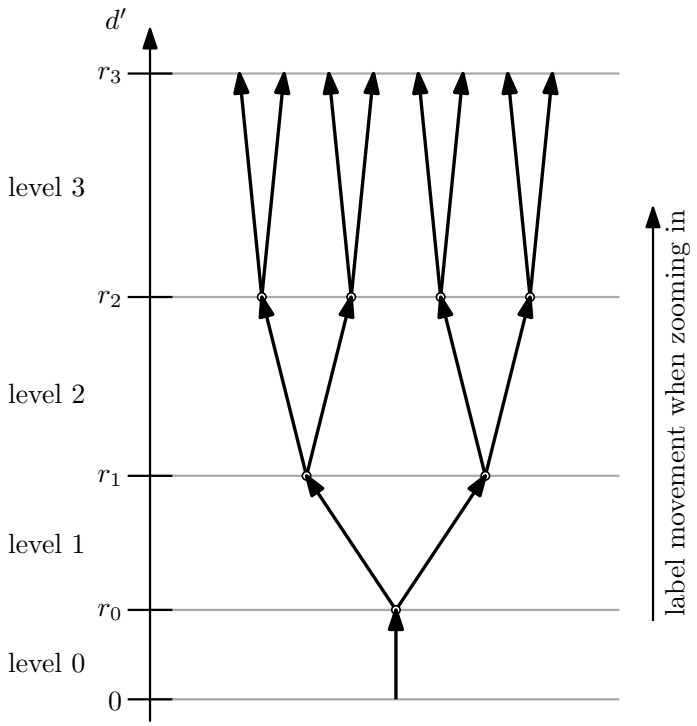


Figure 4.15: The Radial Stabbing Tree now consists of partial of 1D screens.

program. When placing a label in the Radial Stabbing Tree however, the placement does not split the whole instance into two subinstances, but rather splits only one branch of 1D screens in the tree. Hence when now aiming for a dynamic-programing approach, we need one splitting label for each leaf of the tree to split up the whole instance.

Splitting Labels

At first, we consider a stabbing tree with a root node and only two child nodes like the one depicted in Figure 4.16. The first step of the ARO-3D on 1D screen optimal algorithm is to place the lowermost label in the root node: it splits the 1D screen of the root node in two independent sub-instances, as already pointed out in Section 4.2 and depicted in Figure 4.18(a). Since the label's trajectory continues on one of the 1D screens of the child nodes, placing it will also split this child node's 1D screen in two subinstances (Figure 4.18(b)). We call it screen A, and the other child's screen B.

All other labels starting at the root screen and transitioning to screen A are divided into two independent groups and therefore yield two subinstances that can be solved independently. But this is not the case for labels starting in two distinct subinstances of the root screen (Figure 4.19(a)) and then transitioning to screen B (Figure 4.19(b)). Since there is no label trajectory splitting screen B into two subinstances, the two labels might have a conflict at screen B. They cannot be placed independently at the root node's distinct subinstances without risking an invalid solution. This violates the basic assumption that the two subinstances are independent, which is necessary for the dynamic programming approach.

To regain the property, that the two subinstances are completely independent from each other, we have to place a second label b whose trajectory splits screen B in two. Since the root screen is already split by label a, the second splitting label will be placed on screen B only, which means that it will start at the transition from level zero to level one. We will

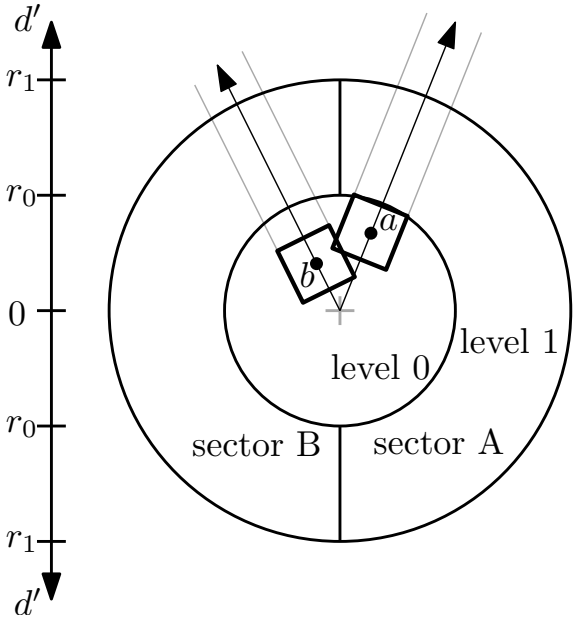


Figure 4.16: The on-screen trajectories of labels embed in the sectors of the Radial Stabbing Tree.

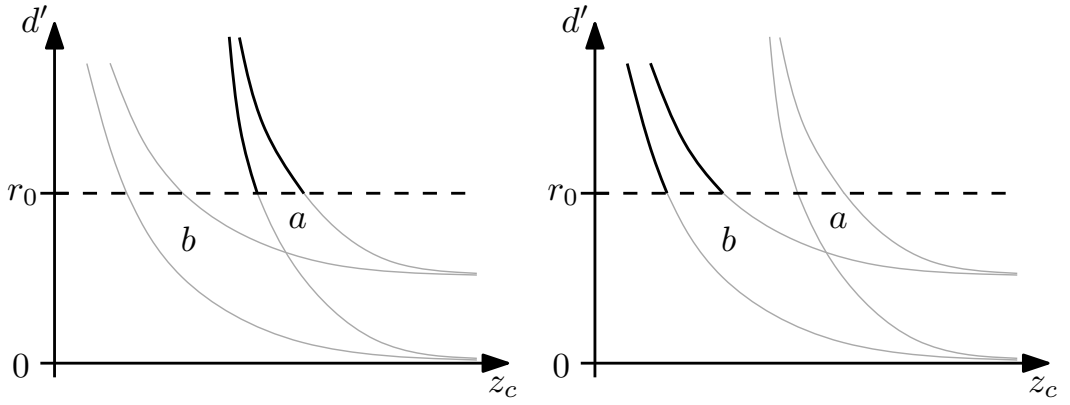
show later on why this does not affect the actual starting point of the label compared to the optimal solution.

The starting point of label b marks a certain point on the transition line between the root sector and the two child sectors, as well as the point where label a transitions from the root sector to child sector A marks such a point. We demand that no other label being placed later on at the root level may cross the level transition between these two points and continue to sector B. That means this segment of the transition line between level zero and level one is part of the border that splits the two subinstances of sector B. This yields two decoupled subinstances, in which further labels can be placed safely without affecting the other subinstance.

When considering a Radial Stabbing Tree with three levels, the additional level (level two) introduces four more sectors to the instance. Labels a and b will continue each in one of the new sectors, since they cannot be switched off in Labelling Model B (see Section 4.1). The other two new sectors are not split into subinstances, so that we have to add another two splitting labels, one for each new "unsplit" sector.

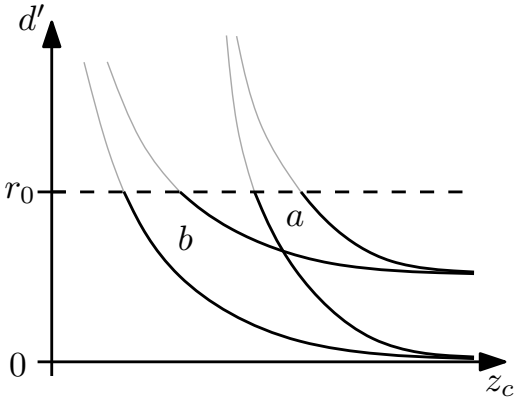
Pushing this observation a bit further, a tree with k levels needs 2^{k-1} splitting labels to be split in two disjoint subinstances on all levels. Hence we have a series of splitting label trajectories that border the two subinstance on one side. Vice versa, the border of a subinstance is given by a starting point of the lowermost label (which is the lower bound of the subinstance on the root level) plus a label index to identify this lowermost label; and then $2^{k-1} - 1$ labels indices that mark the label trajectories splitting the upper sectors.

When it comes to generating the subinstances, we have n possibilities for choosing the lowermost label a. By Lemma 5, a has at most n possible starting points, which means that we choose the first separating label trajectory between two subinstances among n^2 options. Labels splitting the upper sectors are also chosen among maximally n potential labels. Additionally we have to consider the upper and lower bound of z_c as pseudo trajectories splitting a sector. We obtain two special cases as depicted in Figure 4.22. For the upper labels however, the starting point is fix at the bottom of the corresponding sector and not



(a) Level 1, Sector A

(b) Level 1, Sector B



(c) Level 0, Root Sector

Figure 4.17: The label trajectories are split up into disjoint groups when transitioning to a higher level.

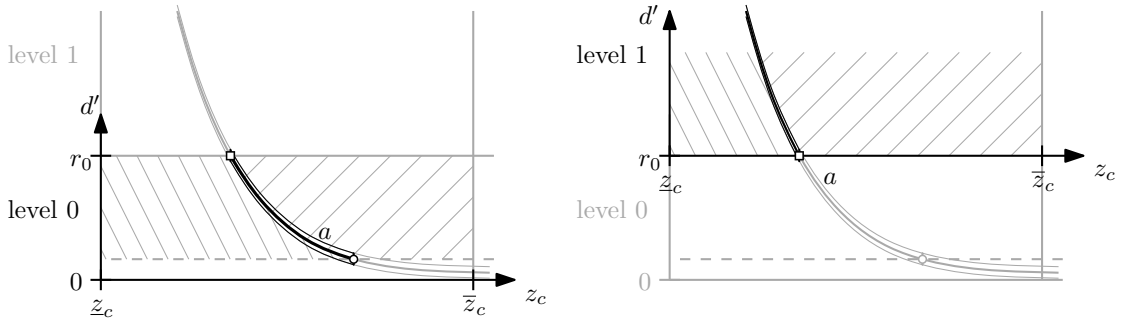
free to be chosen. The border of a subinstance can therefore be completely described by $2 + 2^{k-1} - 1 = 2^{k-1} + 1$ indices $i \in \{1..n+2\}$ (including upper and lower bound of z_c).

A subinstances for the dynamic program presented in Section 4.2 needs a left and a right boundary. Since we can identify one boundary by $2^{k-1} + 1$ indices, a subinstance being identified by two individual borders is therefore completely describable by $2^k + 2$ indices $i \in \{1..n+2\}$.

That means a subinstance is given by a series of left and right bounding label trajectories very similar to the bounding labels presented in Section 4.2 for the dynamic program for the ARO-3D on 1D screen optimal solution. But the main difference is, that the subinstance stretches out over all sectors of the whole tree. We therefore need a left and right label trajectory bounding the subinstance in each sector at each level of the tree. The number of label trajectories necessary to delimit the subinstance doubles with each additional level. That means a subinstance in a tree with k levels needs 2^k splitting labels to be fully qualified. A subinstance in turn can be identified and fully described by its 2^k surrounding label trajectories.

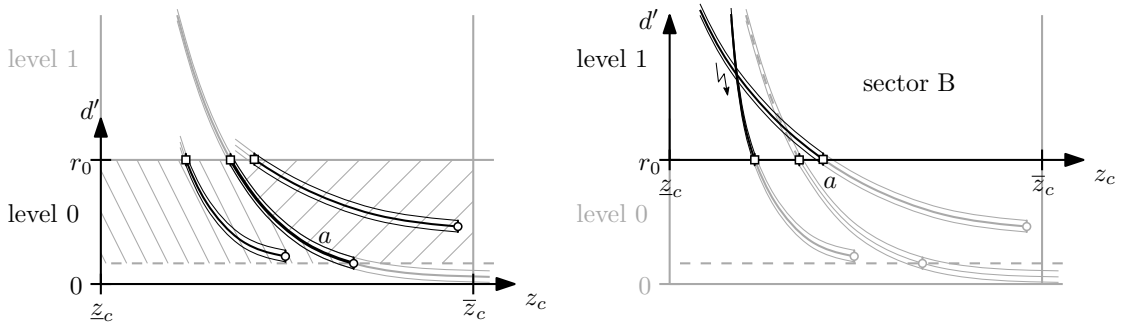
The Dynamic Program

Since we can describe an independent subinstance by 2^k indices in $\{1..n+2\}$ it seems most obvious to model the dynamic program by a table with $(n+2)^{2^k+2}$ entries. Each set of indices represents the series of labels defining the left and right boundary of the label. The entry then contains gain value of the corresponding subinstance with respect to



(a) Level 0, Root Sector (b) Level 1, Sector A

Figure 4.18: Placing a label splits root sector and one of the child sectors.



(a) Level 0, Root Sector (b) Level 1, Sector B

Figure 4.19: Other child sector is not split properly, which can cause a conflict.

the gain function presented in Section 3.1. The table entry is indexed by $2^k + 2$ indices $i \in \{1..n+2\}$.

We split the indices in two groups with each $2^{k-1}+1$ indices, which we call *series*. The first series describes the left boundary of the instance, the second describes the right boundary. The first two indices of a series describe the lowermost label trajectory, which is the starting point of the whole boundary. The first of the two indices refers to the label trajectory, which is to be placed and the second index denotes at which of its conflict events the trajectory starts.

The remaining $2^{k-1} - 1$ indices of a series describe the additional splitting labels necessary to split the upper sectors in two independent instances.

$$T[a, z_a, i_1, .., i_{2^{k-1}}][b, z_b, j_1, .., j_{2^{k-1}}] \qquad , a, z_a, b, z_b, i_*, j_* \in \{1..n+2\} \tag{4.20}$$

To calculate an entry in the table, that means the gain value of one subinstance, we maximize over all possible subdivisions of this subinstance into two separate sub-subinstances. That means we iterate over all possible series of splitting labels contained within this level and calculate the gain that placing those labels at their corresponding levels would contribute. Each series yields two subinstances, which are again represented by other table entries, so that the over all gain of one splitting variant is given by the gain of placing this certain label series plus the gain values of the arising sub-subinstances.

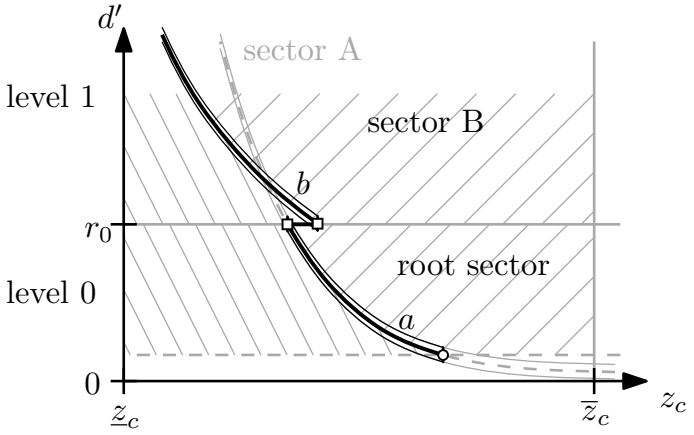


Figure 4.20: Subinstances induced by choosing a second splitting label for the level transition

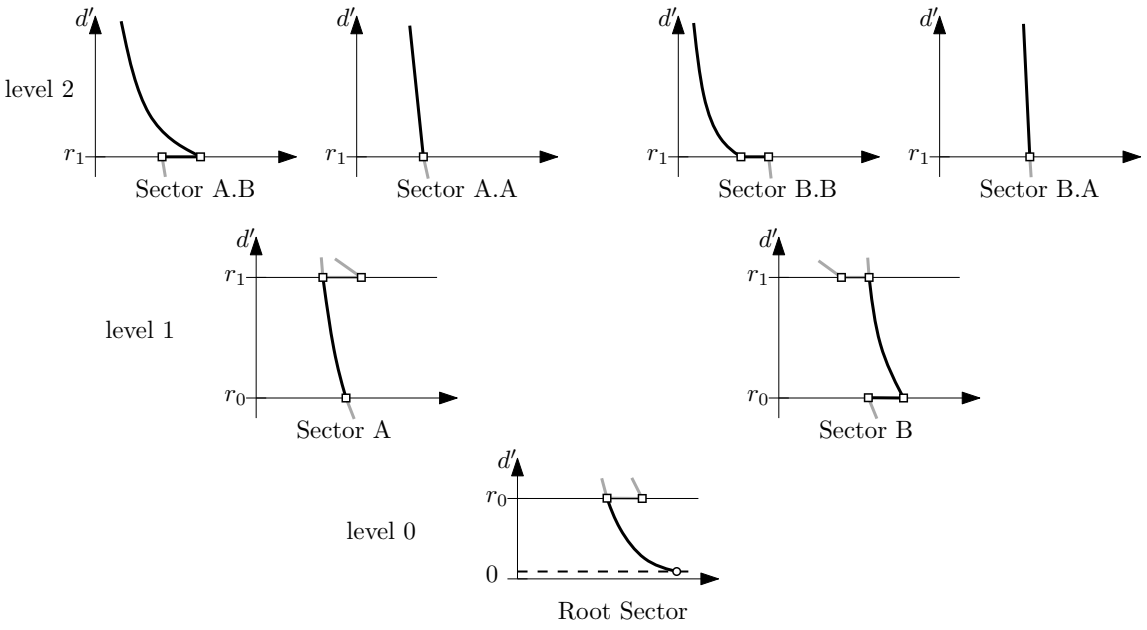


Figure 4.21: The two subinstances spread over three levels of the Radial Stabbing Tree.

$$
\begin{aligned}
&T[a, h_a, i_1, .., i_{2^{k-1}}][b, h_b, j_1, .., j_{2^{k-1}}] \qquad (4.21)\\
&= \max_{[d,h_d,l_1,..,l_{2^{k-1}}] \in \{1..n+2\}^{(2^{k-1}+1)}} \Big\{ g([d, h_d, l_1, .., l_{2^{k-1}}]) + \\
&\qquad T[a, h_a, i_1, .., i_{2^{k-1}}][d, h_d, l_1, .., l_{2^{k-1}}] + T[b, h_b, j_1, .., j_{2^{k-1}}][d, h_d, l_1, .., l_{2^{k-1}}] \Big\}
\end{aligned}
$$

We will not go into detail on how exactly the gain $g([d, z_d, l_1, .., l_{2^{k-1}}])$ of a series is calculated, since this rather a laborious task than theoretically complex, though there are a few technical realisation details to keep in mind.

Technical Details

The lowermost label d of the new series is allowed to be one of the upper splitting labels of the original instance's boundary series:

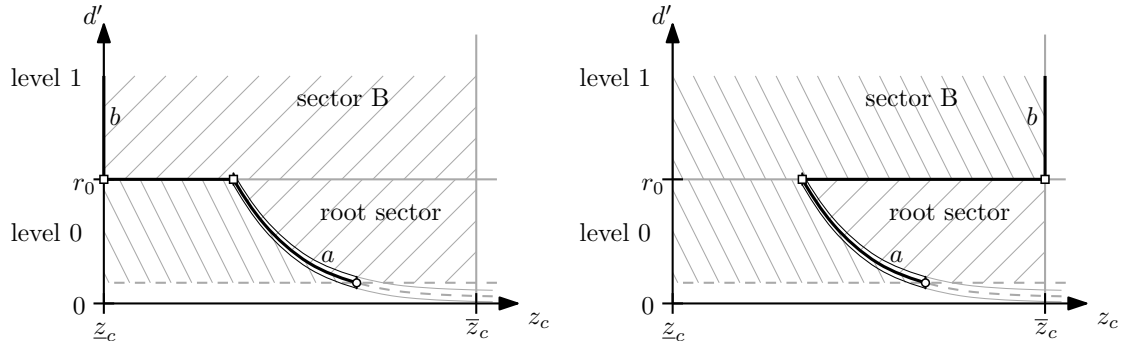
$$d \in \{i_1, .., i_{2^{k-1}}, j_1, .., j_{2^{k-1}}\} \ . \qquad (4.22)$$

The upper splitting labels $l_1, .., l_{2^{k-1}}$ of the new series are allowed to be the same as the upper splitting labels of the original instance's boundary series:

$$l_1, .., l_{2^{k-1}} \in \{i_1, .., i_{2^{k-1}}, j_1, .., j_{2^{k-1}}\} \ . \tag{4.23}$$

The starting point of the lowermost label h_d and the upper splitting labels $l_1, .., l_{2^{k-1}}$ of the new series can also be equal to the upper bound $\overline{z}_c$ or the lower bound $\underline{z}_c$ of the zooming ratio interval, as depicted in Figure 4.22. These boundaries are referred to by the indices $\perp$ and $\top$ respectively, as similarly introduced in Section 4.2:

$$h_d, l_1, .., l_{2^{k-1}} \in \{\perp, \top\} \tag{4.24}$$



(a) Lower bound on zooming ratio $\underline{z}_c$ defines subinstance boundary $\perp$.

(b) Upper bound on zooming ratio $\overline{z}_c$ defines subinstance boundary $\top$.

Figure 4.22: Two special cases of subinstance boundaries define two virtual label indices.

In each of the above cases, it has to be paid special attention that gain values of placed trajectories are not added up twice. That means after when caclulating the gain value $g([d, h_d, l_1, .., l_{2^{k-1}}])$ of the new series, we have to subtract all parts of this series that already were part of the previous boundary series $[a, h_a, i_1, .., i_{2^{k-1}}]$ or $[b, h_b, j_1, .., j_{2^{k-1}}]$.

Additionally, we want to emphasize that label trajectories, which are being placed as part of the new series, always have to be located completely within the current subinstance. This is in accordance with the corresponding formulation for the ARO-3D on 1D screen optimal solution in equation 4.10.

The gain value of the complete instance of the Radial Stabbing tree is then the value of entry

$$T[\perp, \top, \perp, .., \perp][\top, \top, \top, ..., \top] \ . \tag{4.25}$$

Correctness and Completeness

The correctness of the dynamic program is relatively obvious. First, a label switched on as part of placing a series always persists up to the highest level and is not switched off again for any zooming ratio (except when it leaves the screen), which would be illegal in labelling model B.

Secondly, no intersections between label trajectories and therefore no on-screen label intersections are introduced, since new labels are only being placed as part of a series. This series (and all its labels) however, always has to be located completely within the current subinstance. The label trajectories therefore cannot intersect with the subinstance's boundaries or leave the subinstace for any zooming ratio.

The subinstances themselves are always well defined when constructed as presented above. If the concept of additional splitting labels for upper sectors is applied throughout the whole Radial Stabbing Tree, no "gaps" in the subinstances' boundaries can occure and the two subinstaces devided by a series of splitting labels are completely disjoint.

The dynamic program presented above does not lead to any illegal solutions.

For prooving the completeness, we do not show that the algorithm explores all possible configurations, but we rather assume that we knew the optimal solution and show, that this solution is constructed by the above dynamic program. This means that one of the table entries of the dynamic program contains the value of the optimal solution and its indices define the configuration of series leading to this solution.

In the following, we distinguish between *final* placement and *preliminary* placement of a label. A label is placed *finally*, if it is placed as first label d of a series $[d, h_d, l_1, ..., l_{2^{k-1}}]$, which is the currently lowermost label to be placed in an instance. All other labels $l_1, ..., l_{2^{k-1}}$ of this series, which we use to split sectors at higher levels into two subinstances, are placed *preliminarily*. A preliminary starting point $d'_l(\overline{z}_l)$ of a label l therefore always coincides with a level transition

$$d'_l(\overline{z}_l) \in \{r_0, ..., r_{k-1}\} \tag{4.26}$$

whereas a starting point $d'_d(\overline{z}_d)$ of a finally placed label d can lie within a level (and will in general).

$$d'_d(\overline{z}_d) \in [0, r_k] \tag{4.27}$$

A preliminarily placed label will be placed finally in a later iteration.

We already discussed in Section 4.2, that we can unambiguously sort all labels in ascending order by their starting point in the optimal solution on a 1D screen. Now, we sort labels by $d'_d(\overline{z}_d)$, the distance of the labels' starting points from the center of the screen. Since the starting point is equivalent to the label's final placement, we use this order to determine which label is the the next label to be placed finally within an instance.

When thinking of the starting points of labels in the optimal solution as an ordered set, each choice of a series of splitting labels divides this set in two disjoint subsets. Each of the subsets lies completely in the corresponding subinstance and in this subinstance, the next label to be placed finally is the one with the "smallest" starting point with respect to the above defined order.

The next label to be placed when constructing a series $[d, h_d, l_1, ..., l_{2^{k-1}}]$ is l_1, the preliminarily placed label at the transition to the next higher level. This label is splitting the first child sector into two subinstances. That means if the final label d of this series is placed at level i, the preliminary label l_1 is placed at

$$d'_{l_1}(\overline{z}_{l_1}) = r_i \tag{4.28}$$

and splits "the other" child sector at level $i + 1$, that is not split by label d. To gain the optimal solution with the above dynamic program, we choose the label l of the optimal solution that transitions from level i to level $i + 1$ but continues in the sibling sector to the one label d continues in, for which the difference

$$|z_{d,i} - z_{l,i}| \qquad , d'_d(z_{d,i}) = d'_l(z_{l,i}) = r_i \tag{4.29}$$

is minimal (see Figure 4.23).

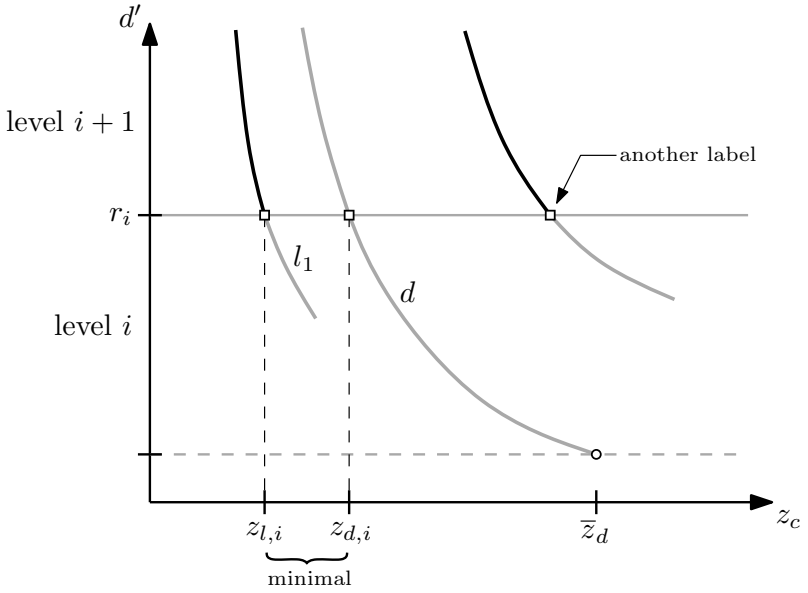


Figure 4.23: The distance between two transitioning labels determines the next preliminary label.

If there is no such label in the optimal solution, we choose the instances boundary as preliminarily placed label l_1. Since then there are no labels transitioning from level i to level $i+1$, it is not import whether we choose the instance's left or right boundary. We proceed in the same manner for the remaining labels to be placed preliminarily.

Putting everything together

All in all, the dynamic program presented in this section does construct the optimal solution and only constructs legal solutions. Hence, this algorithm deterministaically calculates the optimal solution to ARO-3D on a Radial Stabbing Tree of 1D screen sectors.

The optimal solution on a Radial Stabbing Tree of 1D screen sectors however, is a 3-approximation to the optimal solution on the corresponding Radial Stabbing Tree of 2D screen sectors. The optimal solution to this problem in turn is a 2^{k-1}-approximation of the optimal solution to the original ARO-3D on 2D screen problem. This reasoning leads to the following theorem:

Theorem 4. *The optimal solution to the ARO-3D on 2D screen problem in Labelling Model B can be approximated in polynomial time to the constant factor $3 \cdot 2^{k-1}$, where k denotes the number of levels of a Radial Stabbing Tree covering the screen of constant size.*

5. Conclusion

5.1 Results

In this thesis, we investigated on attaching labels to data points in 3D that are presented on a dynamically zoomable viewport.

We introduced basic mathematical methods for displaying 3D data points on a 2D screen, attaching labels to them and characterizing on-screen intersections between the labels. We added certain display dynamics to this model, inasmuch as we investigated on how the data points and labels behave on the 2D screen when zooming in and out. This eventually allowed us to specify different versions of the ARO-3D active range optimization labelling problem.

Based on this problem formulation, we deducted two different labelling models, first in Chapter 3 Labelling Model A, which allows labels to be activated and deactivated at arbitrary zooming ratios but demands maximally one unsplittable activity interval for each label. We proved NP-hardness of ARO-3D on 2D screen for this labelling model by exploiting its close relation to Simple 2d-ARO with proportional delation by Been et al. [BNPW10]. We developed and studied different MILP and ILP formulations (as well as heuristics) and evaluated them quantitatively with respect to performance and quality of the solution.

Secondly, we introduced Labelling Model B, which also allows labels to be activated at arbitrary zooming ratio but demands them to stay active when zooming in further, until they leave the screen. Despite its NP-hardness on 2D screens, we were able to construct an algorithm which calculates an optimal solution on a (theoretic) 1D screen for this problem version of ARO-3D in polynomial time. Based on this result, we deducted a constant factor polynomial time approximation algorithm for ARO-3D on 2D screen in Labelling Model B. For this purpose we introduced the Radial Stabbing Tree as a central novelty, which harnesses the locality property of conflicts between labels.

This thesis is rather to be seen as a first step on a new field of research than being regarded as final work with ready-made solutions. It gives overview over some technical approaches and outlines possible solutions as well as providing tools to reach them. There are many possible directions to advance to from here.

5.2 Future Work

ARO-3D in Labelling Model A is strongly related to the independent set problem, which is NP-hard to approximate to a constant factor. However, there exist versions of the independent set problem, which can be solved approximately in polynomial time. It remains open to investigate, whether ARO-3D in Labelling Model A is not approximable at all or whether maybe one of the special cases of the independent set problem can be exploited to obtain an approximation algorithm for ARO-3D in Labelling Model A.

While working on the results presented in this thesis, we additionally tried several approaches to refine an approximation algorithm for ARO-3D in Labelling Model A. We think that randomized rounding and primal-dual schema approaches to the (M)ILP formulations are very promising for finding an approximation algorithm. In the time given for this thesis, however, we have not acquired final results on these approaches. It is left to future research to further explore these possibilities.

The Radial Stabbing Tree shows great promise for future research. In this work, its usage is very simple and inflexible. We expect it can be used more adaptive to the underlying structure of the labels' trajectories on the screen. It remains to be shown whether this leads to an improvement of the approximation factor. Perhaps it is even possible to find an approximation algorithm independent of the screen to label size ratio or to derive a polynomial time approximation scheme.

There are also different starting points for future research to completely change the labelling model and obtain other label layout problems. For example it can be very useful for some applications to extend the algorithms presented in this work to not only allow for data points to be labelled, but also for line features and whole objects. Additionally one could integrate obstacles into the model, so that data elements or labels that are obstructed by an obstacle may not be displayed.

The way in which labels are attached to their data elements can be changed in various ways. In this thesis, we only regarded one discrete anchor position per label. However, multiple discrete positions or continuous ranges of positions could lead to a higher utilisation of the screen and better results of the labelling layout. Also not attaching labels directly to their data elements, but via line segments, decouples the the placement of the label from the data elements position on the screen

Throughout thesis, we used a constant weighting function to determine the labels' contribution to the gain value of a solution. For some applications other weighting functions might be useful as well. It remains to be examined how this changes the actual label placement problem and solutions to it.

We concentrated our research efforts on labelling dynamics in terms of zooming operations to be applied to the viewport. However, another promising area of future research interest is 3D dynamic labelling with rotation of the viewport. It would be very interesting to incorporate rotation to the 3D dynamic zooming model presented in this thesis.

Bibliography

[AF03] Ronald Azuma and C. Furmanski. Evaluating label placement for augmented reality view management. In *Mixed and Augmented Reality, 2003. Proceedings. The Second IEEE and ACM International Symposium on*, pages 66–75, 2003.

[AHS05] Kamran Ali, Knut Hartmann, and Thomas Strothotte. Label layout for interactive 3d illustrations. *Journal of WSCG.*, 13(1-3):1–8, 2005.

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

[BFH01] Blaine Bell, Steven Feiner, and Tobias Höllerer. View management for virtual and augmented reality. In *ACM Symp. on User Interface Software and Technology*, pages 101–110. ACM Press, 2001.

[BHKN08] Marc Benkert, Herman Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria one-sided boundary labeling. In *Graph Drawing*, volume 4875 of *Lecture Notes in Computer Science*, pages 243–254. Springer Berlin Heidelberg, 2008.

[BNPW10] Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry*, 43(3):312–328, 2010. Special Issue on 24th Annual Symposium on Computational Geometry (SoCG'08).

[dBG12] Mark de Berg and Dirk H.P. Gerrits. Approximation algorithms for free-label maximization. *Computational Geometry*, 45(4):153–168, 2012.

[dBG13] Mark de Berg and Dirk H.P. Gerrits. Labeling moving points with a trade-off between label speed and label overlap. In *Algorithms – ESA 2013*, volume 8125 of *Lecture Notes in Computer Science*, pages 373–384. Springer Berlin Heidelberg, 2013.

[FW91] Michael Formann and Frank 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. ACM, 1991.

[Ger13] Dirk H.P. Gerrits. *Pushing and Pulling*. PhD thesis, Technische Universiteit Eindhoven, 2013.

[GHS06] Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. Agent-based annotation of interactive 3d visualizations. In *Smart Graphics*, volume 4073 of *Lecture Notes in Computer Science*, pages 24–35. Springer Berlin Heidelberg, 2006.

[GHS07] Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. Annotation of animated 3d objects. In *SimVis, SCS*, pages 209–222, 2007.

[GNN13] Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. Trajectory-based dynamic map labeling. In *Proc. 24th Ann. International Symp. on Algorithms and Computing (ISAAC'13)*, volume 8283 of *Lecture Notes in Computer Science*. Springer Verlag, 2013.

[GNR11] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Consistent labeling of rotating maps. *CoRR*, abs/1104.5634, 2011.

[GNS08] Igor Griva, Stephan G. Nash, and Ariela Sofer. *Linear and Nonlinear Optimization (2nd ed.)*. Society for Industrial Mathematics, 2008.

[HAS04] Knut Hartmann, Kamran Ali, and Thomas Strothotte. Floating labels: Applying dynamic potential fields for label layout. In *Smart Graphics*, volume 3031 of *Lecture Notes in Computer Science*, pages 101–113. 2004.

[MD06a] Stefan Maass and Jürgen Döllner. Ein konzept zur dynamischen annotation virtueller 3d-stadtmodelle. In *Kartographische Schriften - Aktuelle Entwicklungen in Geoinformation und Visualisierung (GeoVis)*, volume 10, pages 19–26. Kirschbaum Verlag, Bonn, 2006.

[MD06b] Stefan Maass and Jürgen Döllner. Efficient view management for dynamic annotation placement in virtual landscapes. In *Smart Graphics*, volume 4073 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2006.

[MD08] Stefan Maass and Jürgen Döllner. Seamless integration of labels into interactive virtual 3d environments using parameterized hulls. In *4th International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, pages 33–40. The Eurographics Association, 2008.

[MP09] Konrad Mühler and Bernhard Preim. Automatische annotation medizinischer 2d- und 3d-visualisierungen. In *Bildverarbeitung für die Medizin 2009*, Informatik aktuell, pages 11–15. Springer Berlin Heidelberg, 2009.

[PRS97] Bernhard Preim, Andreas Raab, and Thomas Strothotte. Coherent zooming of illustrations with 3d-graphics and text. In *Proceedings of the Conference on Graphics Interface '97*, pages 105–113, 1997.

[RPRH07] Timo Ropinski, Jörg-Stefan Praßni, Jan Roters, and Klaus Hinrichs. Internal labels as shape cues for medical illustration. In *VMV*, pages 203–212. Aka GmbH, 2007.

[SD08] Thierry Stein and Xavier Décoret. Dynamic label placement for improved interactive exploration. In *Proceedings of the 6th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '08, pages 15–21, 2008.

[SM09] Peter Shirley and Steve Marschner. *Fundamentals of computer graphics*. Peters, Taylor & Francis Ltd., 3. ed. edition, 2009.

[uB10] Ladislav Čmolík and Jiří Bittner. Layout-aware optimization for interactive labeling of 3d models. *Computers & Graphics*, 34(4):378–387, 2010. Procedural Methods in Computer Graphics Illustrative Visualization.

[vKSW99] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Computational Geometry*, 13(1):21–47, 1999.