FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Boundary Labeling for annotated documents

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Bachelor's programme Software & Information Engineering

by

Jakob Klinger
Registration Number 1125755

to the Faculty of Informatics
at the TU Wien

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

Vienna, 25th September, 2017

Jakob Klinger	Martin Nöllenburg

Erklärung zur Verfassung der Arbeit

Jakob Klinger
Scherzergasse 10/8, 1020 Wien

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

Wien, 25. September 2017

Jakob Klinger

Contents

CHAPTER 1

Introduction

Whenever additional information neets to be inserted into an existing document without altering the original text, we can make use of annotations. They usually take the form of footnotes, which require only a minimal reference in the main text, and are used for a variety of reasons - for example, to provide additional information that would hinder the text's flow if inserted directly, or as a result of a commenting tool that is used for communication between an author and their editor. If a more obvious connection between the text and the referenced content is required, for example when lengthy comments are added, or if a change-tracking tool is used, the reference is often placed to the side of the document and visibly connected to the part of the text it is referring to. This style of annotation is easily implemented on virtual documents, since they can be hidden on demand, however if the annotations need to be included in a printed version, there are several issues that arise regarding readability of the final product and ambiguity of text-annotation assignments.

In this thesis, we will look at ways to use Boundary Labeling for this problem, which means that all annotations will be placed somewhere outside of the text they are referencing and will be visually connected to the feature they are referencing. (See also [3])

The guidelines on how to create suitable labelings are as follows: the connections should be as direct as possible, no important information should be obscured, and it should be easily discernable which Label belongs where. These three criteria easily come into conflict with one another, as the text usually is very dense and leaves little space for lines in between, yet one shouldn't allow them to pass through the text, as this makes the text harder to read.

1.1 Terminology and Fundamentals

Boundary Labeling (or equivalent concepts) can be applied to a space with different geometry or more dimensions, but this thesis will only concern itself with two-dimensonal,

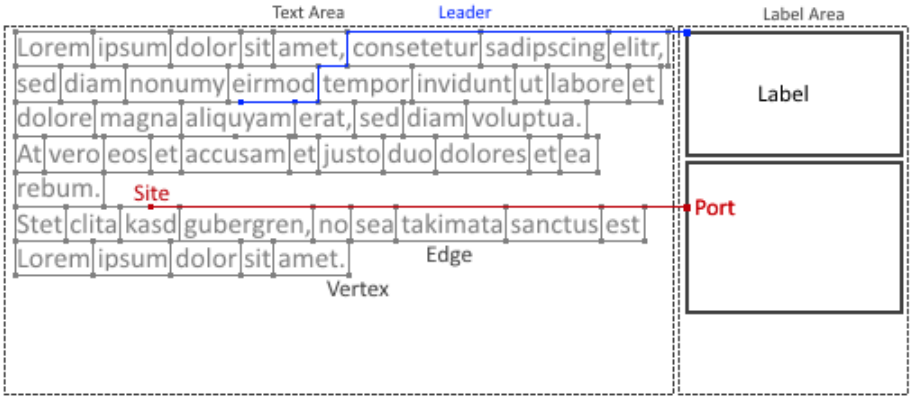

Figure 1.1: Illustrated guide to the labeling terminology

Euclidean space. To easily reference important concepts, some additional terminology will be introduced as well. (See Fig. 1.1 for a visual explanation)

A *graph* $G = \langle V, E \rangle$ is a tuple of *vertices* $V = \{v_1, v_2, ..., v_n\}$ and *edges* $E = \{e_1, e_2, ..., e_m\}$. A vertex v is a featureless object. Each edge e is a relation between two vertices $E \subseteq V \times V$. We call two vertices $u, v \in V$ *adjacent*, if the edge $e = (u, v) \in E$. A *path* $P = v_1, ..., v_h$ is an ordered sequence of vertices, where each vertex must have an edge connecting it to the subesquent one. *Depth-first search* is a searching algorithm on a graph G that starts at a given vertex $v \in V$ and explores the graph by traversing its edges as far as possible before backtracking, and continues to do so until a pre-defined goal is met.

Polylines are a connected series of *line segments*. Line segments are straight lines that contain each point between their starting and end point. *Labels* hold additional information and are represented as boxes containing this information. They are usually placed in the *label area* which is a rectangular area designated to hold labels. It is located next to of the bigger, rectangular *text area*, which contains the document's text and all *sites*, the points or objects that a label's information refers to. If multiple label areas exist on different sides of the text area, we speak of *multi-sided labeling*, otherwise we speak of *one-sided labeling*. We will be using one-sided labeling in our implementation. Some space was left in between the text and label area, to make connecting sites to their labels easier, which we will call the *routing area*. The site and the label are connected via a *leader*, a polyline that can be further classified by looking at the orientation of its segments: *O-Segments* run orthogonally to the border of the label area. *P-Segments* run parallel to the border of the label area, and as such must be combined with other segments for the leader to reach its destination. *S-Segments* are not required to have any particular orientation, and simply connect their start and ending points in a straight line. The leader's name is created by combining the name of the segments - for example, the blue leader from Fig 1.1 would be classified as an OPOPO-Leader. The location where a leader connects to the label is called the *port*. It can be restricted to pre-defined positions.

1.2 Related Work

Boundary labeling was first introduced by Bekos et al. in 2004 (see [3]), where both one-sided and multi-sided labelings with different leader types are looked into. They also showed that the optimal placement of arbitrarily-sized labels on two sides of the text area can be NP-hard by drawing comparisons to the Partition-Problem. However, a pseudo-polynomial solution exists for this problem, which was adapted to this variation of the problem.

Since then, several papers have been written about boundary labeling. One of these is [2], which looks into the readability of different leader styles. Interestingly, some leader styles perform quite well, despite the study's participants preferring others over them, with OPO-Leaders being both least preferred and the hardest to follow.

Another article using boundary labeling is [4] by Göetzelmann et al., which creates boundary labeling-style annotations along other methods to label different parts of three-dimensional figures, resulting in pictures similar to what could be found in a textbook. As this algorithm works in real-time, it is suitable for labeling interactive models and allows for user interaction.

Boundary labeling in text documents however, is rarely discussed, and only few papers exist about this topic. The programs that employ this style of annotation often also use rather simple algorithms, to mediocre results or make extensive use of the interactivity of a digital medium, showing annotations only on demand. However, the few papers that approach this topic add interesting information to the discussion.

The paper about the Luatodonotes-Package[5] uses several styles of drawing leaders, and came to the conclusion that leaders without bends are easier to follow, which fits with [2]'s observations, which ranks OPO- and PO-Leaders lower than other variants. However, most solutions proposed in [5] do not consider whether a path overlaps with text or not, which results in a decrease in readability. While we do not use the routing and leader styles introduced in this paper, the results can be used in comparisons regarding readability of the main text and ease of use of the different leader styles.

The thesis by Loose[7] on the other hand is based around only using the free space between lines and words, which produces longer leaders, and forces curves, but doesn't obscure any part of the text. The two different approaches in this paper were a clustering-based algorithm, which was previously described in [8], and a flow network-based approach. Several concepts of this paper, such as the graph-based strategy and the usage of a routing area will be adopted in our thesis and it is by far the biggest influence on our approach to the problem.

Lin et al.[6] use only OPO-Leaders that have their P-Segment located outside of the text area in their paper, but allow the leaders to use the text area's border on the opposite side of the label area to route upwards or down. This allows for more labels to be placed as close as possible to their leader's source, at the cost of increasing select leaders' length and placing some labels out of order. While this is an interesting way to avoid longer leaders in general, it is hard to combine with the graph-based routing that happens inside the text area, so we won't make use of it.

CHAPTER 2

The Algorithm

2.1 Problem Specification

We limited the leaders to use exclusively O- and P-Segments, and banned them from passing through words. We also place labels as far up as possible to maximize the space remaining for remaining placements. Additionally, a leader isn't allowed to be any longer as is necessary to connect a given Site with its label's port.

These restrictions are implemented as follows: We divided the text T into separate lines $L = \{W, H\}$ which are in turn made up of words W and whitespace H. The remaining space between lines shall be defined as $S(l_1, l_2)$with $l_1, l_2 \in L$ and will have no restrictions regarding leader placement. For each word $w \in W$, we define $R(w)$ as its bounding rectangle, which marks the space leaders aren't allowed to cross. For a graphical representation see Fig. 2.1a.

Since we only allowed the Usage of O- and P-Segments in leaders, the only way for a leader to cross through a line of text is with a P-Segment through whitespace, whereas O-Segments are only usable between lines. Therefore, we can create a routing graph $G = \langle V, E \rangle$ whose Edges E reflect the legal paths a leader can take within T. For each whitespace character $h_x \in W$ in a given line $l_i \in L$, one vertex is placed in $S(l_{i-1}, l_i)$ and $S(l_i, l_{i+1})$ each, located above or below the whitespace character, and with a maximum distance from each of its neighbouring lines. The start and end of each line also recieve a pair of vertices each. Sites will be represented by the insertion of additional vertices in $S(l_i, l_{i-1})$ on a similar height as those next to whitespace of the same line and located directly above the center point of its words w_j bounding rectangle $R(w_j)$. Should either l_{i-1} or l_{i+1} not exist, the vertices' vertical placement will be similar to that of the previous or following line. The edges between the vertices are all perfectly horizontal or vertical, and do neither intersect with any bounding rectangle, nor any vertices other than their starting and ending vertex. The resulting graph looks similar to Fig. 2.1b.

Each edge has a capacity of 1, meaning that no more than one leader is allowed to pass through it. Leaders also aren't allowed to cross one another, as it is hard to discern

Figure 2.1: Visualization of the space reserved for the text, and the resulting graph.

between intersections and two leaders curving away from each other. This is realized by forcing leaders to incorporate as many P-Segments as early as possible while still following all previous restrictions, and forbidding leaders from entering the space to the top left of already successfully routed leader's sites.

2.2 Description

In our Algorithm we first create the routing graph that will be used from here onward. To achieve this, we ge through the annotated text word by word, and measure the length $le(w_i)$ of each word $w_i \in l_j$ that is not part of an annotation to determine its placement in the text. Afterwards, we create new vertices above and below the whitespace characters h_k, h_{k+1} preceding and succeeding w_i. These vertices are located in the space $S(l_{j-1}, l_j), S(l_j, l_{j+1})$ between the word's line and its surrounding lines l_{j-1} and l_{j+1}, as described in the previous section. Whenever an annotation is encountered, a single vertex is added above the last non-annotation word in $S(l_{j-1}, l_j)$ at $pos(h_k) + le(w_i)/2$. For a representation of this process in pseudocode, see Alg. 1.

```
   Data: A text with annotations,stored as a String-Array
   Result: A Graph (as described above)
 1 initialization
 2 foreach w in words do
 3     if w is annotation then
 4         v ←new Vertex(previousWord.getCenter())
 5         v.setAnnotation(w)
 6         Graph.addVertex(v)
 7         UpperVerticesList.addVertex(v)
 8     else
 9         if w is too big for the line then
10             startNewLine()
11             connectBasedOnPosition(UpperVerticesList)
12             UpperVerticesList ←LowerVerticesList
13             emptyList(LowerVerticesList)
14         end
15         v1 ←new Vertex(w.getTopLeft())
16         v2 ←new Vertex(w.getTopRight())
17         v3 ←new Vertex(w.getBottomLeft())
18         v4 ←new Vertex(w.getBottomRight())
19
20         Graph.addAll(v1,v2,v3,v4)
21         UpperVerticesList.addAll(v1,v2)
22         LowerVerticesList.addAll(v3,v4)
23         Graph.createEdgeBetween(v1,v3)
24         Graph.createEdgeBetween(v2,v4)
25     end
26 end
```

Algorithm 1: Representation of the Graph-creation algorithm in pseudocode

For the routing part of the algorithm, we work through the annotations $a \in A$ in the order they appear in the text, placing each as far up as possible, skipping any annotation that cannot be placed above its leader's source node. We will also use fixed ports located in the top left corner of each label. This allows us to place annotations by only knowing their port location, which can be inferred from their leader's ending point. The routing area between the text and label areas will be used to bridge the gap between the highest reachable point within the routing graph and the highest possible label placement by connecting the two via an OPO-leader.

The algorithm's input consists of the Graph G and a set of sites $V_{ann} \subset V$ which will be routed using a depth-first search algorithm, restricted to only selecting vertices located above or to the right of the current vertex v_{curr}, prioritizing moving up whenever possible. After the algorithm terminates, it returns a set with routing information for

each vertex, which contains a Path $P = \{v_1, \cdots, v_n\}$, leading from the site to the text area's border, along with information on how to draw the OPO-Segment that connects to the Label's port. If the routing for any given site failed, the path consists of a single vertex - the site. (For an illustration in Pseudocode see Alg. 2.)

```
   Data: A single annotation's source and its Graph
   Result: A List of vertices describing the Leader's Path
 1 initialization
 2 while currentVertex not at right text border do
 3     if (Graph.getTopNeighbourOf(currentVertex) ≠ null) ∧ ¬backtracking
        then
 4         Path.addVertex(currentVertex)
 5         currentVertex ← Graph.getTopNeighbourOf(currentVertex)
 6
 7     else if Graph.getRightNeighbourOf(currentVertex) ≠ null then
 8         Path.addVertex(currentVertex)
 9         currentVertex ← Graph.getTopNeighbourOf(currentVertex)
10         backtracking ← False
11     else
12         backtracking ← True
13         repeat
14             oldVertex ← currentVertex
15             currentVertex ← Path.getLastEntry()
16             Path.RemoveVertex(currentVertex)
17         until currentVertex's Position is below oldVertex or Path is Empty
18
19         if currentVertex not below oldVertex then //No path found
20             break
21         end
22     end
23 end
```

Algorithm 2: The Depth-First-Search algorithm used in the program.

CHAPTER 3

Implementation

The program was written in Java 1.8.0u40, using JGraphT1.0.1[1] as graph library. Since we only want to create leaders that don't intersect with the text, the graph was created alongside the placement of the words on the canvas.

3.1 Challenges

CHAPTER 4

Evaluation and Testing

4.1 Data generation

4.2 Testing methods

4.3 Results

CHAPTER 5

Conclusion

5.1 Further notes

Bibliography

[1] Jgrapht - a free java graph library.

[2] Lukas Barth, Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. *On the Readability of Boundary Labeling*, pages 515–527. Springer International Publishing, Cham, 2015.

[3] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. *Boundary Labeling: Models and Efficient Algorithms for Rectangular Maps*, pages 49–59. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[4] Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. *Agent-Based Annotation of Interactive 3D Visualizations*, pages 24–35. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[5] Philipp Kindermann, Fabian Lipp, and Alexander Wolff. *Luatodonotes: Boundary Labeling for Annotations in Texts*, pages 76–88. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[6] C. C. Lin, H. Y. Wu, and H. C. Yen. Boundary labeling in text annotation. In *2009 13th International Conference Information Visualisation*, pages 110–115, July 2009.

[7] Amrei Loose. Annotation von texten unter berücksichtigung von textzwischenräumen. Master's thesis, Karlsruhe Institute of Technology, 2015.

[8] Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 310–319, New York, NY, USA, 2010. ACM.