# Boundary Labeling for annotated documents

Jakob Klinger
Matr.: 1125755
Curriculum: 033 534
e1125755@student.tuwien.ac.at

September 10, 2017

## 1 Introduction

Annotating a document is usually solved by adding footnotes or figures in an appropriate position and adding a simple reference in the text, leaving the reader to find the referenced content by themselves. Sometimes however, if a more obvious connection between the text and the referenced content is required, the reference is visibly connected to the text by drawing a straight line or a more complex path between them.

In this paper, we will look at ways to use Boundary Labeling, which means that all annotations will be placed outside of the text they are referencing and will be visually connected to the feature they are referencing. (See also [2])

The guidelines on how to create an optimal labeling 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 to which site. These three easily come into conflict with one another, especially when labeling documents, 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.

While there are many papers discussing Boundary Labeling in general, only very few exist that apply this concept to written text. Generally, this approach isn't used very often, and tends to use simplistic algorithms which produce mediocre results. However, the papers that do discuss boundary labeling in text offer interesting contributions.

For example, the paper about the Luatodonotes-Package[3] illustrates some of the different styles of drawing these connecting paths, and came to the conclusion that paths without bends are easier to follow. However, most solutions proposed in that paper don't care whether a path overlapped with text or not, which results in a decrease in readability.

The paper by Loose[4] on the other hand is based around only using the free space between lines and words, which produces longer paths, and forces curves, but doesn't obscure any part of the text.

## 2 Terminology and Fundamentals

While Boundary Labeling (or an equivalent concept) can be applied to a space with different geometry or more dimensions, this paper will only concern itself with two-dimensonal, euclidean space. To easily reference important concepts, some additional terminology will be introduced as well. (See Fig. 1 for a visual explanation)

- **Graph:** A Set of Vertices and Edges (see below), often used to represent routing problems. Can be further classified depending on its properties.
- **Edge:** A connection between two vertices, usually represented as a line. Can be directional, limiting its usage, or have a cost ("Weight") associated with its use.

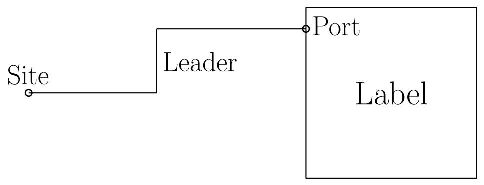

Figure 1: Illustrated guide to the labeling terminology

- **Vertex:** A featureless object that can be connected to any number other vertices via edges. Usually depicted as a point or small shape. Also referred to as **"Node"**.
- **Path:** An ordered series of either edges that share one vertex each with the edges listed before and after them, or an ordered series of vertices that have an edge connecting each vertex to the subsequent one. Also known as **"Graph Walk"**.
- **Polyline:** A connected series of straight lines.
- **Label:** Additional information, usually represented as a box containing the additional information. Will also be referred to as **"Annotation"** in this paper.
- **Site:** The point, object or word that is labeled - without it, there wouldn't be any labeling necessary.
- **Leader:** The line or polyline connecting the site to the label - depending on its shape, it can be described further using several subgroups which can be freely combined to form new leader types:
  - *S-Leader:* This Leader connects site and label in a straight line.
  - *O-Leader:* This leader runs orthogonally to the border between text and label area.
  - *P-Leader:* This Leader runs parallel to the border between text and label area. It must be combined with other leader styles, as it won't reach the label area otherwise.

  For example, the leader from Fig. 1 would be classified as an opo-Leader.
- **Port:** The location where the leader connects to the label. It may be restricted to pre-determined locations, like only at the corners.
- **Monotonous/Monotonicity:** Describes a steady progression towards a direction or goal. Will be used in this paper to indicate that there's no loss of progress made from the site to the label, effectively restricting the direction a leader can go at each bend.
- **Depth-first search:** A searching algorithm that continues along a set of rules as far as possible until either the goal is found, or backtracking is required. A common example is navigating a maze by turning the same way whenever possible.

## 3 The program

In our program, the focus was put on keeping the text as readable as possible. This means that leaders are only allowed in the space between words or lines, which can make certain locations for sites unavailable, as there might already too many leaders routed through that area to insert yet another for the new site. We also wanted the leaders to be monotonous, which means that they will be one of the shortest possible connection between port and site locations. Our final priority was to efficiently use the space for the labels, which meant that each label should initially be placed as close to the available area's border as possible. While this allows for more space in which labels can be placed, it also means that the distance between Label and site often increases, which leads to longer leaders.

### 3.1 Algorithm

To easily create leaders that exclusively use the space between words and lines, we decided to use a graph similar to the one Loose [4] used, placing vertices above and below each word's beginning and end, and connecting them with their horizontally and vertically adjacent neighbours.For the sites, we inserted an additional vertex above the center of the word, which will serve as the leader's starting point. We also decided to give the graph's edges a capacity of 1, which means that each leader will always use edges which haven't already be taken by other leaders. While this causes some sites to become un-routable due to another leader passing right through their starting vertex, it makes visualizing the leaders much easier.

Our first approach was going through the labels in the order they appear in the text and placing each as far up as possible, to maximize the remaining space other labels could use. We also opted to use fixed ports located in the top left corner of the annotation, as this allowed us to unambiguously place any annotation by knowing either their port's location, wich is also the leader's ending point. However, since putting the labels right next to the text would result in the labels' positions being restricted by the text's line spacing, we then decided to leave a buffer zone in between the two areas, which allows the leaders to adjust between the position forced by the line spacing and the label's final position. As this allowed us to place the labels more freely, we adjusted the label's positions after all labels were placed to reduce the distance that needed to be covered in the buffer area.

### 3.2 Implementation

The program was written in Java, using only JGraphT[1] as additional 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, as it was easy to extract measurements at this point. Due to some problems with Java's various methods of calculating linebreaks, this process was done completely by hand.

As the Graph's vertices represent fixed locations on the canvas, they each have coordinates associated with them, with the vertices representing the sites containing extra information, such as references to the corresponding annotation and to the leader connecting the two (if existent). This additional information is used for both routing and drawing the leaders: The routing was done via a depth-first search algorithm that prioritized vertices located further above if possible, and vertices to the right otherwise. After the algorithm terminates, it returns a vertex-based graph walk describing the route from the site to the border of the text area alongside information about where to draw the OPO-segment in the buffer zone if successful, or a walk containing only the starting vertex, if no valid path was found. Using this information alongside the positional data stored in the vertices, the polyline representing the leader can be easily drawn.

## References

[1] Jgrapht - a free java graph library.

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

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

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