A Smart Algorithm for Column Chart Labeling

Sebastian Müller[1] and Arno Schödl[2]

[1] Institut für Informatik, Humboldt-Universität, 10099 Berlin, Germany
sebastian@muellerdigital.net
[2] think-cell Software, 10115 Berlin, Germany
aschoedl@think-cell.com

Abstract. This paper presents a smart algorithm for labeling column charts and their derivatives. To efficiently solve the problem, we separate it into two sub-problems. We first present a geometric algorithm to solve the problem of finding a good labeling for the labels of a single column, given that some other columns have already been labeled. We then present a strategy for finding a good order in which columns should be labeled, which repeatedly uses the first algorithm for some limited look-ahead. The presented algorithm is being used in a commercial product to label charts, and has shown in practice to produce satisfactory results.

1 Introduction

Column charts in all their variations are among the most common forms of data visualization. The need for an automated solution arises when charts are frequently updated and manually placed labels have to be repeatedly rearranged. So far, standard commercial software does not offer automatic and intelligent chart labeling.

In the research community, different areas of automatic layout problems have been considered [1]. Cartographic labeling problems of point, line and area features have traditionally received the most attention. Most variants of these labeling problems are NP-hard [2] [3] [4]. In particular, for point features, various approaches have been tried, among them gradient descent [5], rule-based systems [6] [7], simulated annealing [8] and treating it as a combinatorial optimization problem [9] [10]. Typically, the optimization criterion is either the number of labels, which can be placed without collisions, or the maximum font size for which all labels can still be placed. Alternatively, if labels are allowed to be positioned away from their features and connected by a line, minimizing the length of connectors is a good goal function [11]. A set of constraints forbids label-label and label-point intersections. More recently, several rule-based algorithms for the point feature labeling problem have been developed which prune impossible or redundant solutions from the solution space and then search the remaining solutions with greater efficiency [12] [13]. There also exist approximative algorithms guaranteed to run in polynomial time [14] [15].

Unfortunately, in practice, applying general point feature labeling to column chart labeling gives unsatisfactory results, and no specialized algorithms have

been published. The number of labels and their size is usually set by the user, and must be respected by the algorithm. To be aesthetically pleasing, the solution must look orderly, which rules out the typical label clouds generated by point feature algorithms. Finally, the solution needs to be computed at interactive speed, for example to be integrated into a commercial presentation software like PowerPoint.

For each segment to be labeled, a few labeling positions must be considered. If there is enough space, the label should be put into the column segment. When the label is only slightly too large to fit into the segment, putting the label into a little box of background or segment color can increase legibility. If the label collides with labels of segments above or below, labels can be horizontally staggered. To avoid further collisions, some labels can be put to the side of the column if the space between columns is wide enough. Finally, for very wide labels or in case of small column spacing, labels can be arranged above or below their columns.

Although our implementation considers all possible positions described above, this paper focuses on the final, and most difficult placement of stacking labels above or below their columns. For an orderly appearance, we arrange the labels belonging to one column in a stack, where labels are in the same order as their corresponding segments. Each stack can have its connectors on the left or right side, which poses a combinatorial problem (Fig. 1 (d)). When adding a placement quality metric, the problem turns into an optimization problem. The solution is constrained by disallowing label-label and label-segment intersections.

2 Problem Definition

As the name implies, a column chart is made of a number of columns which are composed of multiple segments. Each segment has a label and some of these labels must be placed as a block on top of the column. In addition, each column can have a sum label which must be placed directly on top of the block of segment labels but can be moved independently in the horizontal direction. The problem is to find placements for all labels on top of their columns, and decide for each block of labels if it should be right- or left-aligned, with the goal of minimizing the total height of the chart with its labels. The following constraints which are illustrated in Fig. 1 must be observed:

a) On the aligned side, the block labels cannot be moved over the column edge to leave room for connector lines to be drawn.
b) The invidual labels must not intersect other labels and can only intrude other columns as long as they do not intersect horizontal segment boundaries. Non-intruding solutions are preferred over intruding ones if they are otherwise of equal quality. Allowing segment intersections here may seem odd, but we found it to significantly improve appearance.
c) The sum label is always placed on top of the column and other labels.

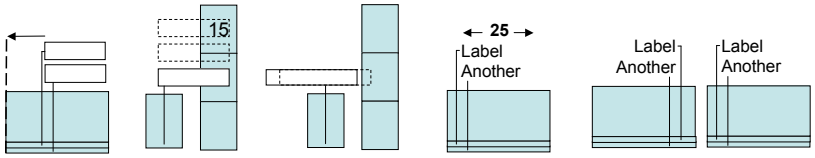

Fig. 1. The different types of choices and constraints: (a) Labels cannot be moved over the column edge on the aligned side; (b.1) Labels can intersect neighboring columns but not segment bounds or labels; (b.2) if possible, a solution which does not intersect a neighboring column is preferred; (c) the sum label is always placed on top and can be moved independently in the horizontal direction; (d) Labels can be right- or left-aligned;

3 Finding a Local Solution

To make the labeling problem tractable, we separate it into two sub-problems. The first is finding the best placement of a single block of labels belonging to one column, given that some blocks of labels of other columns have already been placed. Given such an algorithm, the second problem is a strategy in which order columns should be processed. We start by describing an algorithm for the first, more geometric problem.

In order to find the best placement of a block of labels, collisions with other, already placed label blocks and the chart segments themselves must be avoided. More specifically, we must compute the best 2D position, represented by a shift vector $\boldsymbol{V}$ relative to the optimal position of the label block right above the column. This vector is computed by procedure CalculateBestPosition given the label block, the set of all labels and, implicitly, the chart segments. As a quality criterion for CalculateBestPosition we are using the distance of the label block from its desired position right above the column it belongs to.

A *frontier* is a structure which provides an efficient way of finding this optimal position. It is essentially a function defined over a set of geometrical shapes S. This function can be defined as $f(x) = \max\{\ y \mid (x, y) \in\ s \wedge s \in S\}$. That means a frontier only contains the maxima in y-direction of all shapes contained in S. The function $f(x)$ and the shapes in S are represented as piece-wise linear approximations. The frontier provides the two operations Add and Distance which allow adding a shape to S and computing the distance between a given shape and the frontier, respectively. Of course, we can similarly define frontiers for the other three directions (Fig. 2).

A trivial way to compute the position of a label block would be to create a frontier containing the outline of the chart itself and of all already placed labels and to let the block of labels fall down at a certain x-coordinate. However, using this strategy places the labels on top of the first obstacle they encounter, even if there is sufficient space below this obstacle to fit the label. This space cannot

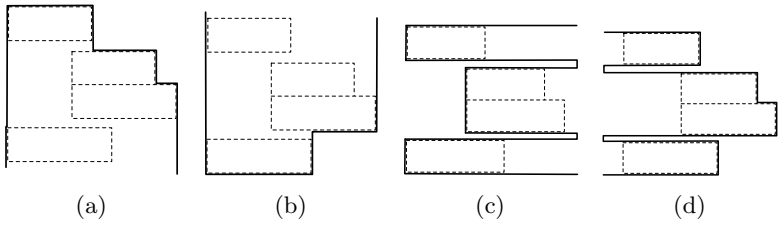

Fig. 2. Possible frontiers: (a) vertical orientation, growing to the left, (b) vertical orientation, growing to the right, (c) horizontal orientation growing down (d) horizontal orientation growing up

be modelled by a vertically oriented frontier. However, we can use horizontally oriented frontiers to look for a label position between the boundaries of neighboring columns and other labels: a left one named F_l growing towards higher x-coordinates and F_r, the right frontier growing towards the lower x-coordinates. Both approaches are compared in Fig. 3. Then, we have to devise an efficient way to find a space between those bounds which is wide enough to fit the labels and which is closest to the desired label position immediately above the column.

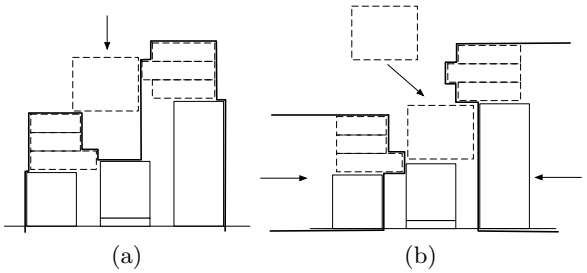

Fig. 3. Calculating the label block position: (a) letting the labels fall down vertically; (b) letting the labels slide into their position between horizontal frontiers resulting in a much better solution.

The function MoveOverFrontier shown on page 5 moves a list of shapes C over frontier F and computes a function $g(x)$ which for every x returns the maximum y so that moving C by (x, y) will make C touch but not intersect the frontier F. For the left frontier F_l, which can be defined as

$$F_\mathrm{l}(y) = \max\{\ x \mid (x, y) \in S\}, \tag{1}$$

the function MoveOverFrontier would compute the function

$$g(y) = \max\{\ x \mid \forall\ (x', y') \in C \wedge F_\mathrm{l}(y' + y) \geq x' + x\}. \tag{2}$$

As all shapes in C, frontier F and function $g(y)$ are represented as piecewise linear approximations, we can compare every shape in C and the frontier F line segment by line segment. Every line of every shape is moved over every line segment of F. Depending on the frontier segment, two cases have to be distinguished (l. 5 and l. 10). In both cases we calculate two vectors, one which moves the line along the frontier segment and another which moves the line over the end of the frontier segments. We can now regard these vectors as simple line segments and add them to our new frontier $\overline{F}$. In a regular frontier F, for every position y, F describes the maximum x-coordinate of all contained shapes. In the newly formed frontier $\overline{F}$, for every y, $\overline{F}$ describes the x-coordinate which makes the shapes in C touch but not intersect F.

Algorithm 1: MoveOverFrontier Algorithm

Input: A list of shapes C and a frontier F
Output: A frontier $\overline{F}$ containing the vectors which move all shapes in C along F

```
 1 frontier F̄;
 2 foreach shape ∈ C do
 3     foreach line ∈ shape do
 4         foreach lineFrontier ∈ F do
 5             if lineFrontier.from.x ≤ lineFrontier.to.x then
 6                 ptFrom = line.to - lineFrontier.to;
 7                 ptTo = line.to - lineFrontier.from;
 8                 F̄.Add( Line(ptFrom, ptTo));
 9                 F̄.Add( Line(ptTo, ptFrom - (line.to - line.from), ptFrom));
10             else
11                 ptFrom = line.from - lineFrontier.from;
12                 ptTo = line.to - lineFrontier.from;
13                 F̄.Add( Line(ptFrom, ptTo));
14                 F̄.Add( Line(
15                 ptFrom - ( lineFrontier.to - lineFrontier.from ), ptFrom));
16             end
17         end
18     end
19 end
20 return F̄;
```

Using the frontier and the MoveOverFrontier algorithm we can now implement the procedure CalculateBestPosition as follows. We calculate for every label block which has to be placed the two frontiers F_l and F_r representing the rightmost and leftmost bounds of the chart's parts to the left or the right of the labels' column.

Then we create the frontiers $\overline{F_\mathrm{l}}$ and $\overline{F_\mathrm{r}}$ by moving our label block L over F_l and F_r. The frontiers $\overline{F_\mathrm{l}}$ and $\overline{F_\mathrm{r}}$ define the space of possible solutions for the

label block which is available between F_l and F_r. For every move by a given y, moving the label block by the resulting $\overline{F_\mathrm{l}}(y)$ or $\overline{F_\mathrm{r}}(y)$ will make it touch the frontiers F_l or F_r, respectively. If for a given y, $\overline{F_\mathrm{r}}(y) > \overline{F_\mathrm{l}}(y)$, then there is not enough space between F_l and F_r at position y to fit the label.

Because the sum label is allowed to move in horizontal direction independently of the block of segment labels, we repeat the same procedure for the sum label, thereby creating two more frontiers $\overline{F_\mathrm{l}}'$ and $\overline{F_\mathrm{r}}'$ from F_l and F_r.

We then iterate over the four frontiers $\overline{F_\mathrm{l}}, \overline{F_\mathrm{r}}, \overline{F_\mathrm{l}}', \overline{F_\mathrm{r}}'$ at once. The 4-tuple of line segments $(s_\mathrm{l}, s_\mathrm{r}, s'_\mathrm{l}, s'_\mathrm{r}) \in \overline{F_\mathrm{l}} \times \overline{F_\mathrm{r}} \times \overline{F_\mathrm{l}}' \times \overline{F_\mathrm{r}}'$ defines a part of our solution space. We subdivide segments as necessary so that all segments $(s_\mathrm{l}, s_\mathrm{r}, s'_\mathrm{l}, s'_\mathrm{r})$ have the same start and end y-coordinates. In this area we search for a shift vector $\boldsymbol{V}$ which is closest to our preferred initial position, ie. closest to a shift $(0, 0)$, and a vector specifying the sum label position $\boldsymbol{V'}$. $\boldsymbol{V}$ and $\boldsymbol{V'}$ are constrained to share the same y-coordinate, but may have different x-coordinates, reflecting independent horizontal movement of the sum label.

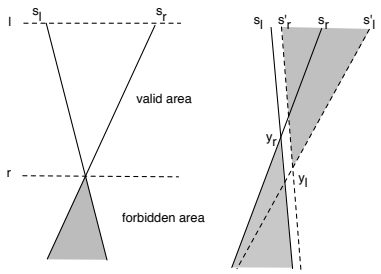

Fig. 4. (a) The two frontier segments s_l and s_r are shown as an example. y_l and y_r define the limits of the solution space. (b) Considering all four frontier segments the both pairs can intersect in a way, that our solution space is empty and $y_\mathrm{l} > y_\mathrm{r}$ holds.

We find the solution $\boldsymbol{V}$ for the label block in the space between the segments s_l and s_r. The sum label solution $\boldsymbol{V'}$ is in the space defined by s'_l and s'_r. Intersections between the segments (cf. Fig. 4) indicate that there is no room at this position to fit the labels. Let $[l, r]$ be the interval which limits the space of valid solutions between s_l and s_r and let $[l', r']$ be the interval which limits the space of valid solutions between s'_l and s'_r. We calculate the left and right boundary of solution space $y_\mathrm{l} = \max(l, l')$ and $y_\mathrm{r} = \min(r, r')$. If $y_\mathrm{l} > y_\mathrm{r}$, our solution space is empty because we have two disjunct solution spaces for the label block and the sum label. If the segments did not intersect, it is still possible that the solution space is empty because the segments overlap in the whole interval. If $y_\mathrm{l} < y_\mathrm{r}$, we shorten all four segments $s_\mathrm{l}, s_\mathrm{r}, s'_\mathrm{l}, s'_\mathrm{r}$ to the interval $[y_\mathrm{l}, y_\mathrm{r}]$.

Now, the solution $\boldsymbol{V}$ is the point closest to $P = (0, 0)$ and point P can either be inside the polygon defined by s_l and s_r or the solution is the point on the polygon outline closest to P which is easily computed by projecting the point on all four line segments of the rectangle's outline and by choosing the point closest to P of the four solutions obtained. The sum label solution $\boldsymbol{V'}$ with the same x-coordinate as $\boldsymbol{V}$ is then guaranteed to exist because the solution space is not empty and it is easily found between s'_l and s'_r.

All of the above is actually done twice, for the label block aligned on the left and on the right. We then choose the alignment with a position closer to $(0, 0)$.

4 Determining the Labeling Order

After having explained how a single label block can be placed, the second, more strategic problem of determining a labeling order remains. A simple approach is to iterate simultaneously from the left and right over the set of all columns, labeling the left labels right-aligned and the right labels left-aligned. At each step of the iteration we place the left or right label block, whichever has the better placement. This approach guarantees that all label blocks can actually be placed without collisions: When proceeding on the left side, we use right-aligned label blocks, which have their connecting lines on the right, and which can only intersect labels to their left, which have already been placed. Collisions with these labels can be avoided by placing the new label block high enough. The same holds for the right side. The advantage of this simple approach is that it always yields a solution, the disadvantage, however, is that the solution will often have the form of a pyramid with labels stacked on top of each other towards the center. To improve the algorithm, we can order the columns by their height and label them starting with the lowest column. Unfortunately, placing label blocks in an arbitrary order can prevent a column to be labeled at all, if all room above the column is taken up by other labels. We avoid this dead-end by inserting an artificial shape above each unlabeled column, blocking all space above the columns, as illustrated in Fig. 5. Although the average results of this variant are much better, there is still an easily identifiable worst-case example. If the columns increase in height from the left to the right, the labels will also be stacked one on top of the other.

To avoid this problem, instead of predetermining the order of labeling, at each step, we calculate the best label block position for each column, given the already placed labels. We again avoid the dead-end described above by blocking the space above unlabeled columns. After calculating the best possible positions for each column, we choose the column with the lowest top label block border to be the one to place its block at its calculated position. The rationale behind this criterion is to free as much room above columns as possible as early as possible, to give more space to future label placements.

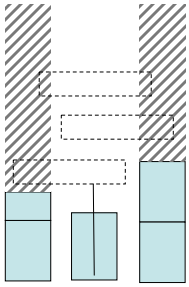

Fig. 5. The look-ahead of the MultiFrontierLookAhead algorithm: the left and right column which have not yet been labeled are blocked in order to guarantee that it can still be labeled in the future. The middle column cannot be labeled in this step because the label is too wide.

We found that in many cases, this heuristic ordering is actually close to the order that a human would use to place labels. The algorithm is guaranteed to find a solution, which in the worst case deteriorates to the simple pyramid of stacked labels described in the beginning of this section.

5 Extensions

One constraint has been ignored in the solution so far. Figure 1 shows labels which overlap their neighboring columns, which is allowed, as long as they do not overlap contained labels or segment boundaries. The geometric algorithm makes the solution easy and it has been omitted to facilitate the presentation. When in procedure CalculateBestPosition the Frontiers F_l and F_r are created, we can add the horizontal segment and column bounds and the contained labels to the Frontier, and not the segments themselves which thus may be intruded. We can compute both solutions, with and without intruding, and pick the non-intruding one if it is otherwise no worse than the intruding one. Likewise, other aesthetic constraints can be easily included into our formulation. For example, if labels should have a margin, we can inflate the shapes added to the Frontier by a small amount.

To obtain interactive speed, we use two efficiency optimizations. Line 25 of the MultiFrontierLookAhead algorithm already shows that after placing a label l_opt, only the labels affected by placing l_opt must be recalculated. Notice that they may not only be affected by collisions with the newly placed label, but also by the freeing of the blocked space above the column belonging to l_opt. Secondly, we can exploit the fact that the Frontiers F_l and F_r in CalculateBestPosition can be calculated recursively: We can compute F_l^n for column n by copying F_l^{n-1} from column $n-1$ and adding the shapes of the next column. After placing a new label block, only the affected range of Frontiers must be recalculated.

Algorithm 2: MultiFrontierLookAhead Algorithm

```
1  L ← list of label blocks;
2  foreach l ∈ L do
3  |  l.top ← highest y-value of the label block's outline;
4  |  l.labeled ← false;
5  |  l.hasvalidsolution ← false;
6  end
7  while ∃ l ∈ L : l.labeled = false do
8  |  l_opt ← nil;
9  |  V_opt ← nil;
10 |  foreach l ∈ E do
11 |  |  if l.labeled = false then
12 |  |  |  if l.hasvalidsolution = false then
13 |  |  |  |  V = CalculateBestPosition(l, L);
14 |  |  |  |  if V is valid then
15 |  |  |  |  |  l.hasvalidsolution ← true;
16 |  |  |  |  end
17 |  |  |  end
18 |  |  |  if l.hasvalidsolution = true
   |  |  |  ∧ (l_opt = nil ∨ l.top + V.y < l_opt.top + V_opt.y) then
19 |  |  |  |  l_opt ← l;
20 |  |  |  |  V_opt ← V;
21 |  |  |  end
22 |  |  end
23 |  end
24 |  PlaceLabel (l_opt, V_opt);
25 |  l.hasvalidsolution ← false for all labels l which intersect with l_opt
26 end
```

6 Evaluation

The worst case for our greedy algorithm is a column chart in form of an inverted pyramid where the columns are getting successively lower towards the middle (Fig. 6 (a)). If all labels are too wide to fit over their column the algorithm will start labeling them from the left- and rightmost column effectively creating a pyramid of stacked labels mirroring the pyramid of columns. Under the constraints specified in the problem definition this is the best labeling. A human user would possibly try to find a solution which violates as few constraints as possible. However, this case can typically be resolved by making the chart a little bit wider. This example is also not typical for a column chart because all labels have the same width and all labels are wider than their respective columns. Figure 6 (b) shows a more typical representative of column charts which is labeled optimally. The positive and negative columns are treated as separate problems and the negative values are labeled downwards. In the second column the value

3.540 is stacked on top whereas in the fourth column it is not. In the second column the value 100 has to be moved to the top because it is too large and intersects a big portion of the segment below. As a result, the 3.540 is moved to the top too. Otherwise, there would not be enough space to fit the connecting line in the segment without intersecting the label 3.540. Case (c) is an extreme example which shows, that the algorithm always finds a solution even when the chart becomes very small.

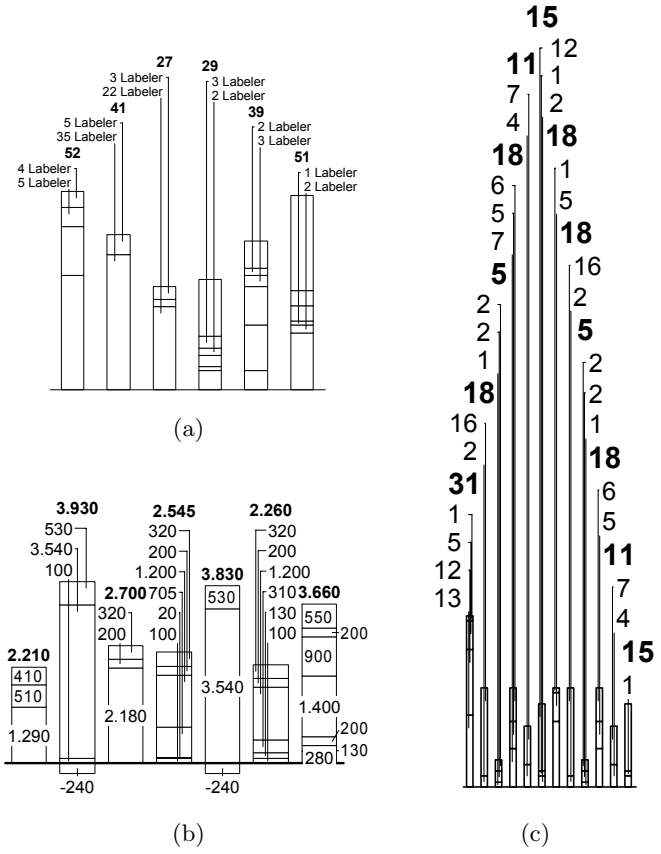

Fig. 6. Labeling examples with timing information. Timing tests were made on an Athlon 64 3800+ machine: (a) Worst-case example where all labels are wider than their respective columns (Labeling took 60.5 ms); (b) Typical case with an optimal solution (115 ms); (c) Extreme case with a very small chart (157 ms)

7 Conclusion

The presented algorithm has been implemented as part of the commercial charting software think-cell chart, and customers, comparing it to manually labeled charts, are satisfied with its performance. Even for worst-case examples, the algorithm provides solutions which pass as good enough in practice. In general, we found that for practical applications worst-case performance is more important than the average performance metrics often used in academic papers.

8 Acknowledgements

This work has been funded by think-cell Software GmbH. We want to thank Prof. Dr. Hans-Dieter Burkhard and Dr. Gabriele Lindemann from the Dept. of Artificial Intelligence at Humboldt-University, Berlin.

References

[1] Steven Lok and Steven Feiner. A survey of automated layout techniques for information presentations. *Proceedings of SmartGraphics, Hawethorne, USA*, 2001.

[2] J. Marks and S. Shieber. The computational complexity of cartographic label placement. Advanced Research in Computing Technology TR-05-91, Harvard University, 1991.

[3] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. *Proceedings of the 7th Annual ACM Symposium on Computational Geometry*, pages 281 – 288, 1991.

[4] C. Iturriaga and A. Lubiw. Np-hardness of some map labeling problems. Technical Report CS-97-18, University of Waterloo, 1997.

[5] S. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5 – 17, 1982.

[6] J. S. Doerschler and H. Freeman. A rule-based system for dense-map name placement. *Communications of the ACM*, 34(1):68 – 79, 1992.

[7] A. C. Cook and C. B. Jones. A prolog rule-based system for cartographic name placement. *Computer Graphics Forum*, 9(2):109 – 126, 1990.

[8] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 14(3):203 – 232, 1995.

[9] R.G. Cromley. An lp relaxation procedure for annotating point features using interactive graphics. *Proceedings of Auto-Carto 7*, pages 127 – 132, 1985.

[10] S. Zoraster. The solution of large 0-1 integer programming problems encountered in automated cartography. *Operations Research*, 38(5):752 – 759, 1990.

[11] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. In János Pach, editor, *Proceedings of 12th Int. Symposium on Graph Drawing*, volume 3383 of Lecture Notes in Computer Science, pages 49 – 59. Springer Verlag, 2005.

[12] A. Wolff. *Automated Label Placement in Theory and Practice*. PhD thesis, Freie Universität Berlin, 1999.

[13] F. Wagner, A. Wolff, V. Kapoor, and T. Strijk. Three rules suffice for good label placement. *Algorithmica Special Issue on GIS*, 2000.

[14] Pankaj K. Agarwal, Marc van Kreveld, and Subash Suri. Label placement by maximum independent set in rectangles. *Proceedings of the 9th Canadian Conference on Computational Geometry*, pages 233 – 238, 1997.

[15] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13:21 – 47, 1999.