Boundary-Labeling Algorithms for Panorama Images

Andreas Gemsa
Karlsruhe Institute of Technology
gemsa@kit.edu

Jan-Henrik Haunert
Universität Würzburg
jan.haunert@uni-wuerzburg.de

Martin Nöllenburg
Karlsruhe Institute of Technology
noellenburg@kit.edu

ABSTRACT

Boundary labeling deals with placing annotations for objects in an image on the boundary of that image. This problem occurs frequently in situations where placing labels directly in the image is impossible or produces too much visual clutter. Previous algorithmic results for boundary labeling consider a single layer of labels along some or all sides of a rectangular image. If, however, the number of labels is large or labels are too long, multiple layers of labels are needed.

In this paper we study boundary labeling for panorama images, where n points in a rectangle R are to be annotated by disjoint unit-height rectangular labels placed above R in k different rows (or layers). Each point is connected to its label by a vertical leader that does not intersect any other label. We present polynomial-time algorithms based on dynamic programming that either minimize the number of rows to place all n labels, or maximize the number (or total weight) of labels that can be placed in k rows for a given integer k. For weighted labels, the problem is shown to be (weakly) NP-hard, and we give a pseudo-polynomial algorithm to maximize the weight of the selected labels. We have implemented our algorithms; the experimental results show that solutions for realistically-sized instances are computed instantaneously.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling

General Terms

Algorithms

Keywords

GIS, visualization, panorama images, label placement, boundary labeling, sliding labels, dynamic programming

Manuscript submitted to ACM GIS 2011.

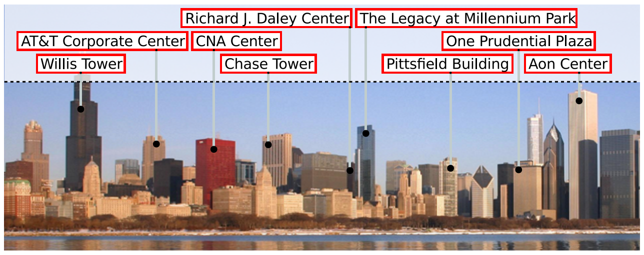


Figure 1: Panorama labeling for the skyline of Chicago. Photography: ©J. Crocker.

1. INTRODUCTION

Annotating features of interest in images by textual labels or icons is an essential aspect of information visualization. Depending on application and image content these labels are either placed directly next to the features within the image or, if partial occlusion of the image by labels is unacceptable or if feature density is too high, on the image boundary. In the first, so-called *internal labeling model*, which is common, e.g., for placing object names in topographic maps, the association between feature and label should be clear from the spatial proximity between them. This is no longer the case in the latter *boundary labeling model* and hence features and associated labels are connected to each other using simple arcs. In this paper we consider a new and practically important boundary labeling variant, motivated by labeling features in panorama images of, e.g., skylines or mountain ranges. Such labeled illustrations are frequently used for describing landmarks and buildings at lookout points or in tourist guide books. Moreover, very wide panorama photographs of streets of houses as used in popular commercial digital road maps often shall be annotated by information about local businesses or other points of interest. A common aesthetic requirement in all these examples is that the labels are placed above a *horizon line*, e.g., in the area taken up by the sky or simply above the actual image.

In this paper we present efficient algorithms for computing optimal boundary labelings of such panorama images. Figure 1 shows a labeling in our model: we are given a set of n anchor points (or *sites*) in a rectangle R and for each point a variable-width but unit-height open rectangular label (think of the bounding box of the object name written as a single line of text). In order to achieve at least a horizontal proximity between points and labels, every label must be placed vertically above its associated anchor point. Each label and its associated site are connected by a vertical line segment, denoted as a *leader*. Our labeling model, in which the lower edge of a label can slide horizontally along the upper leader endpoint, is a so-called *one-slider* model (in contrast to fixed-position models). The algorithmic problem is to select a subset of the labels and for each selected label compute a label position (i.e., a row index and a horizontal slider position) such that no two labels overlap and no leader intersects any label other than its own. We consider two basic optimization goals: (i) minimize the number of rows required to place *all* labels, and (ii) maximize the number of labels that can be placed in k rows.

We start with a review of related work on boundary labeling and point out our main contributions.

1.1 Related Work

Algorithmic label placement problems have been studied in computational geometry for more than 20 years now [6]; a vast body of literature is collected in the map-labeling bibliography [14]. Most of the literature, however, is concerned with internal label placement as traditionally used in cartography. Boundary labeling as an alternative labeling model was first introduced as an algorithmic problem by Bekos et al. [4] and has subsequently been studied in various flavors, see also the recent survey by Kaufmann [10].

Different boundary labeling models can be classified by (a) the shape of the leaders, (b) at which sides of R labels can be placed, and (c) further restrictions about the labels such as variable or uniform size, placement in multiple layers etc. Leaders are usually represented as polygonal lines; for readability, the leader shape should be as simple as possible. Leaders of arbitrary orientation without bends are called *straight* or *type-s* leaders. To reduce visual clutter axis-aligned leaders are often preferred over arbitrary type-s leaders. The shape of an axis-aligned polygonal leader starting from the anchor point is described by a string over the alphabet $\{p, o\}$, where p and o denote, respectively, leader segments parallel and orthogonal to the side of R containing the label. If a segment is diagonal at a fixed angle (e.g., 45°), we use the letter d to refer to its orientation.

Bekos et al. [4] presented efficient labeling algorithms in the one-, two-, and four-sided model using type-s, type-*po* and type-*opo* leaders. Their main objective was to minimize the total leader length, but they also presented an algorithm for minimizing the number of bends in one-sided *opo*-labeling. Benkert et al. [5] studied algorithms for one- and two-sided *po*- and *do*-labeling with arbitrary leader-dependent cost functions (including total length and number of bends); the algorithms were implemented and evaluated experimentally. Bekos et al. [1] presented algorithms for combinations of more general *octilinear* leaders of types *do*, *od*, and *pd* and labels on one, two, and four sides of R. For uniform labels the algorithms are polynomial, whereas the authors showed NP-completeness for a variant involving non-uniform labels.

Extensions of the basic static point-feature model include algorithms for labeling area features [3], and a dynamic one-sided *po*-model, in which the user can zoom and pan the map view while the label size on the boundary remains fixed [12]. Relaxing the property that each label is connected to a unique site leads to the many-to-one model. In this model NP-completeness results, approximations and heuristics for crossing minimization with type-*opo* and -*po* leaders are known [9]; Lin [11] presented an approach using duplicate labels and hyperleaders to avoid crossings.

The only previous work using multiple layers of labels on the boundary presented $O(n^4 \log H)$-time algorithms for label size maximization in a one-sided model with two or three "stacks" of labels on a vertical side of R and type-*opo* leaders [2] (here H is the height of the rectangle R). In the algorithms all labels are assumed to be of uniform size and a maximum scaling factor is determined such that all labels can be placed in the available stacks. The authors further gave NP-hardness results for some two-stack variants of non-uniform labels and *opo*- or *po*-leaders.

1.2 Contribution

In our paper we study a one-sided multi-row labeling problem with type-o leaders and variable-width labels. Note that for comparison with the results of Bekos et al. [2], the same model can be transformed into an equivalent multi-stack labeling problem with variable-height labels; since for textual annotation variable-width labels are more relevant, we describe the multi-row model. In Sect. 2 we introduce our labeling model in more detail and define three optimization problems: *MinRow*, which aims to find a labeling with all n labels in the minimum feasible number k^* of rows, *MaxLabels*, which maximizes the number of labels placed in k given rows, and *MaxWeight*, which considers labels weighted by importance (with total weight w_{total}) and computes a maximum-weight subset of labels for k rows. Section 3 describes an $O(k^* n^3)$-time algorithm for *MinRow*, an $O(kn^3 w_{\text{total}}^2)$-time algorithm for *MaxWeight*, and an $O(kn^3)$-time algorithm for *MaxLabels*. In Sect. 4 we present extensions of the algorithms for practically interesting variations of the basic problems. We have implemented our algorithms and report results of an experimental evaluation in Sect. 5.

2. PANORAMA LABELING MODEL

We aim to label a set $P = \{p_1, \ldots, p_n\}$ of points, where $p_i = (x_i, y_i)$. We assume $x_i < x_{i+1}$ for $i = 1, ..., n-1$, i.e., the points $p_1, \ldots, p_n$ have distinct x-coordinates and they are ordered from left to right. Moreover, we require $y_i < 0$ for $i = 1, \ldots, n$ meaning that p_i lies below the horizontal line $y = 0$, which we denote as the *horizon*. For our problems, the y-coordinates of the points in P are irrelevant and we can assume that all points lie on the horizontal line $y = -1$. For each $p_i \in P$ we are given an open axis-parallel rectangular label L_i of width $W_i \in \mathbb{R}_0^+$ and height 1.

Labeling P comprises two subproblems: selecting the labels that are to be displayed and placing them onto the plane. More formally, we define a *(panorama) labeling* as a set $S \subseteq \{L_1, \ldots, L_n\}$ and a point (X_i, Y_i) for each $L_i \in S$, where $X_i \in \mathbb{R}$ is the x-coordinate of the right boundary of L_i and $Y_i \in \mathbb{Z}^+$ is the y-coordinate of the lower boundary of L_i. By requiring $Y_i > 0$ for all labels we ensure that all labels are placed above the horizon. Restricting Y_i to integer values allows us to say that label L_i is placed in *row* Y_i.

In order to connect each label $L_i \in S$ with its corresponding point $p_i \in P$, we draw a vertical line segment from (x_i, Y_i) to (x_i, y_i); we call this line segment the *leader* of label L_i and point p_i.

A labeling is *feasible* if it satisfies requirements (F1)–(F3):

(F1) For every label $L_i \in S$ the leader of L_i actually connects L_i with p_i, i.e., $X_i - W_i \leq x_i \leq X_i$.

(F2) For every label $L \in S$ the leader of L does not intersect any label in S other than L.

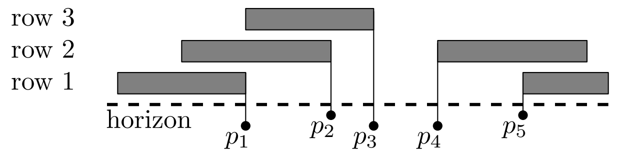


Figure 2: A feasible labeling using $\lceil n/2 \rceil = 3$ rows.

(F3) The labels in S do not intersect each other.

For a pair of indices $1 \le i \le j \le n$ and a row index k, we define an (i,j,k)-*labeling* as a feasible labeling of $P_{ij} = \{p_i, \dots, p_j\}$ with $S = \{L_i, \dots, L_j\}$ satisfying

(R1) $Y_i = Y_j = k$, i.e., both L_i and L_j are in row k, and

(R2) $Y_\ell \le k$ for $\ell = i+1, ..., j-1$, i.e., the labels for all points between p_i and p_j are in row k or below.

We say that an (i,j,k)-labeling $\mathcal{L}$ is *compact* if there is no (i,j,k)-labeling where label L_j has a smaller x-coordinate than in $\mathcal{L}$; we denote the x-coordinate of L_j in a compact (i,j,k)-labeling $\mathcal{L}$ as the *x-value* $\mathcal{X}_{i,j,k}$ of $\mathcal{L}$. If no (i,j,k)-labeling exists we set $\mathcal{X}_{i,j,k} = \infty$.

Among the feasible labelings we can optimize different objective criteria. Generally, we try to avoid wasting space and we favor solutions that display many labels. A basic problem is to find a solution that selects all labels and uses as few rows as possible.

Problem 1 (MinRow). *Given a set P of n points below the horizon and a label for each point, find a feasible labeling with* all *labels that requires the minimum number of rows.*

In fact, a feasible labeling with all labels always exists.

Lemma 1. *For each MinRow instance there exists a feasible labeling with $\lceil n/2 \rceil$ rows and for each $n \in \mathbb{N}$ there exists a MinRow instance with n points that requires $\lceil n/2 \rceil$ rows.*

Proof. To show the first part of Lemma 1, we label all points in P as in Fig. 2, i.e., for $i = 1, \dots, \lceil n/2 \rceil$ we set $Y_i = i$ and $X_i = x_i$; for $i = \lceil n/2 \rceil + 1, ..., n$ we set $Y_i = n - i + 1$ and $X_i = x_i + W_i$. Clearly, requirement (F1) holds. Requirement (F2) holds since for $i = 1, ..., n-1$ and $j = i+1, ...n$ label L_j lies above Y_i if $j \le \lceil n/2 \rceil$ and to the right of x_i if $j > \lceil n/2 \rceil$. Requirement (F3) holds since each row either contains a single label or two labels L_i, L_j with $i \le \lceil n/2 \rceil < j$. In the latter case L_i lies to the left of L_j since the right boundary of L_i is $X_i = x_i$, the left boundary of L_j is $X_j - W_j = x_j$, and $i < j$ implies $x_i < x_j$. To show that $\lceil n/2 \rceil$ rows may be required let $p_i = (i, -1)$ and $W_i = n$ for $i = 1, ..., n$. Since the distance between p_1 and p_n is $n-1$ at most two labels fit in one row. □

Obviously, solutions as the one in Fig. 2 are rather useless in practice—placing no more than two labels in a single row the space consumption becomes huge. If the horizontal distances between the points are small and the labels are wide, however, we will fail to find any better solution. Therefore, we allow some labels to be discarded if the available space is limited. If we are restricted to a certain number of rows, then we want to display as many labels as possible.

Problem 2 (MaxLabels). *Given a set P of n points below the horizon, a label for each point, and a positive integer K, determine the feasible labeling that displays the maximum number of labels in at most K rows.*

By counting the number of labels we might fail to measure the quality of a labeling appropriately. We should account for the fact that often some objects are more important than others and thus deserve a higher chance of becoming labeled. This can be expressed with *weighted* points.

Problem 3 (MaxWeight). *Given a set P of n points below the horizon, a label for each point, a positive integer K, and a positive integer weight w_i for each point $p_i \in P$, find the feasible labeling S that maximizes the weight sum $\sum_{L_i \in S} w_i$ among all feasible labelings that use at most K rows.*

3. ALGORITHMS

3.1 Single-Row Optimization

Before we describe our algorithm for MinRow in the next subsection, we briefly discuss the following single-row label placement problem, which serves as a base case for MinRow.

Problem 4 (SingleRow). *Given a set P of n points below the horizon and a label for each point, decide whether there is a feasible labeling for P with* all *labels in a single row above the horizon.*

The SingleRow problem is closely related to a single-machine job scheduling problem, where n ordered jobs $J_1 < \dots < J_n$ with processing times p_i and release and due times r_i and d_i are to be non-preemptively scheduled in the given order such that all jobs finish before their due time. The weighted version of this problem is known as single-machine throughput maximization and has been related to one-dimensional weighted point labeling problems before [13].

SingleRow can be solved with a simple greedy strategy that processes the points in increasing x-order and places the next label L_i in the leftmost possible position such that it does not intersect the previous label, i.e., $X_i = \max\{X_{i-1} + W_i, x_i\}$. If for any i we have $X_i > x_i + W_i$, then obviously no feasible single-row labeling exists (requirement (F1) violated) and we report failure and return ∞. Otherwise we report success and return the position X_n of the last label. The correctness of the algorithm is immediate and for sorted points it takes linear time.

With the same algorithm we can compute the x-values $\mathcal{X}_{i,j,1}$ of all compact $(i,j,1)$-labelings in $O(n^2)$ time by running it n times, once for every subset $P_{i,n}$ $(i = 1, \dots, n)$ of points and storing in each step $j = i, \dots, n$ of the algorithm with a feasible labeling the position of label L_j (or ∞ otherwise) as the x-value $\mathcal{X}_{i,j,1}$.

3.2 Row Number Minimization

We now show how to solve MinRow efficiently using dynamic programming. Our idea is to construct compact (i,j,k)-labelings for all $1 \le i \le j \le n$ and successively increasing values of k until a feasible labeling for all points is found.

To ease notation we introduce two dummy points p_0 and p_{n+1} to the left of p_1 and to the right of p_n that do not influence the feasibility of labeling P, regardless of the position of L_0 and L_{n+1}. We set $W_0 = W_{n+1} = 0$, $x_0 = x_1 - 2W_{\max}$, and $x_{n+1} = x_n + 2W_{\max}$, where $W_{\max} = \max_{1 \le i \le n} W_i$ is the maximum label width.

Our algorithm computes a three-dimensional table $\mathcal{T}$, where each entry $\mathcal{T}[i,j,k]$ for $0 \le i \le j \le n+1$ and $1 \le k \le \lceil n/2 \rceil$ stores the x-value $\mathcal{X}_{i,j,k}$ of a compact (i,j,k)-labeling as well as some backtracking information in order to reconstruct the compact (i,j,k)-labeling. With this semantics it is clear that there is a solution to MinRow with k rows if and only if there is a feasible $(0,n+1,k)$-labeling, i.e., $\mathcal{T}[0,n+1,k] < \infty$.

We compute $\mathcal{T}$ in a bottom-up fashion with respect to the row index k, i.e., we first compute $\mathcal{T}[\cdot,\cdot,k]$ for $k=1$ and if $\mathcal{T}[0,n+1,1]=\infty$ proceed to computing $\mathcal{T}[\cdot,\cdot,k]$ for $k=2$ and so on until eventually $\mathcal{T}[0,n+1,k]<\infty$. The entries of $\mathcal{T}[\cdot,\cdot,k]$ for $k=1$ are computed by the algorithm described in Sect. 3.1. For $k>1$ the entries $\mathcal{T}[i,i,k]$ for $i=0,\dots,n+1$ are set to $\mathcal{T}[i,i,k]=\mathcal{X}_{i,i,k}=x_i$. We use the recurrence $\mathcal{T}[i,j,k]=\min\theta^k_{i,j}$ for all $i<j$ and $k>1$, where $\theta^k_{i,j}$ for $i<j$ and $k>1$ is defined as the set

$$\theta^k_{i,j} = \left\{ \max\{x_j, \mathcal{T}[i,\ell,k]+W_j\} \;\middle|\; \begin{array}{c} i \le \ell < j, \\ \mathcal{T}[i,\ell,k] \le x_j, \\ \mathcal{T}[\ell,j,k-1] < \infty \end{array} \right\}.$$

Note that $\theta^k_{i,j}$ can be empty; in that case we define $\min\emptyset := \inf\emptyset = \infty$ and obtain $\mathcal{T}[i,j,k]=\infty$. Algorithm 1 summarizes the dynamic program.

```
1 compute $\mathcal{T}[\cdot,\cdot,1]$ using the greedy single-row algorithm
2 for $k = 2$ to $\lceil n/2 \rceil$ do
3     for $j = 1$ to $n+1$ do
4         for $i = 0$ to $j-1$ do
5             $\mathcal{T}[i,j,k] = \min \theta^k_{i,j}$
6         $\mathcal{T}[j,j,k] = x_j$
7     if $\mathcal{T}[0,n+1,k] < \infty$ then return $k$
```

Algorithm 1: Bottom-up computation of table $\mathcal{T}$.

We now prove that this approach is efficient and correct.

Theorem 1. *Algorithm 1 solves MinRow in $O(k^* n^3)$ time, where k^* is the number of rows in the optimal MinRow solution.*

Proof. The running time of Algorithm 1 follows immediately. The outer loop is iterated k^* times, and in each iteration of the outer loop we perform $O(n^2)$ iterations of the nested inner loops. In each iteration the most time consuming task is to find in linear time the minimum of the set $\theta^k_{i,j}$, which contains $O(n)$ elements.

It remains to prove the correctness of Algorithm 1 by showing $\mathcal{T}[i,j,k]=\mathcal{X}_{i,j,k}$ for all $0\le i\le j\le n+1$ and $1\le k\le k^*$. The proof is by induction over k and for each k by induction over $j-i$.

For $k=1$ we use the trivial greedy algorithm of Sect. 3.1 and it follows that the entries $\mathcal{T}[\cdot,\cdot,1]$ are correct.

Next we prove correctness for $k>1$. In the base case $i=j$ the algorithm sets $\mathcal{T}[i,i,k]=x_i$, which is the x-value $\mathcal{X}_{i,i,k}$ of a compact (i,i,k)-labeling that simply consists of the single label L_i placed in its leftmost possible position $X_i=x_i$ in row $Y_i=k$.

Now let's assume $i<j$ and let $\mathcal{L}$ be an (i,j,k)-labeling. By definition L_i and L_j are in row k. This implies that there is a well-defined *predecessor* L_ℓ of L_j in row k for some $i\le\ell<j$; we also say that L_ℓ *precedes* L_j in row k. We call a label L_ℓ a *feasible* predecessor of L_j if there exists an (i,j,k)-labeling, where L_ℓ precedes L_j. Let $F(i,j,k)$ be the set of all feasible predecessors of L_j in an (i,j,k)-labeling. Now we define an (i,j,k)-labeling $\mathcal{L}$ to be *ℓ-compact* if L_ℓ precedes L_j and there is no other (i,j,k)-labeling with predecessor

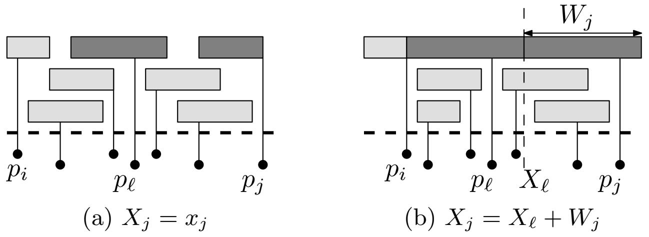


Figure 3: Two ℓ-compact $(i,j,3)$-labelings.

L_ℓ where the position of L_j is further to the left. Figure 3 shows two ℓ-compact $(i,j,3)$-labelings.

Obviously, every compact (i,j,k)-labeling $\mathcal{L}$ is also ℓ-compact for the predecessor L_ℓ of L_j in $\mathcal{L}$. On the other hand the ℓ-compact (i,j,k)-labeling with the leftmost position of L_j over all feasible predecessors $L_\ell\in F(i,j,k)$ is a compact (i,j,k)-labeling. For every ℓ-compact (i,j,k)-labeling the leftmost x-coordinate of L_j is $X^\ell_j=\max\{x_j, X_\ell+W_j\}$, since L_j must be W_j to the right of its predecessor but cannot be left of x_j, see Fig. 3. Hence the x-value of a compact (i,j,k)-labeling $\mathcal{X}_{i,j,k}=\min\{X^\ell_j \mid L_\ell\in F(i,j,k)\}$. Note that if $F(i,j,k)=\emptyset$ we obtain $\mathcal{X}_{i,j,k}=\infty$. We claim that $\theta^k_{i,j}=\{X^\ell_j \mid L_\ell\in F(i,j,k)\}$ and thus Algorithm 1 correctly computes $\mathcal{T}[i,j,k]=\min\theta^k_{i,j}=\mathcal{X}_{i,j,k}$.

Let $\mathcal{L}$ be an ℓ-compact (i,j,k)-labeling and let $\mathcal{L}_{\text{left}}$ be the (i,ℓ,k)-labeling formed by labels $L_i,\dots,L_\ell$ of $\mathcal{L}$. We can assume that $\mathcal{L}_{\text{left}}$ is compact since otherwise we can replace $\mathcal{L}_{\text{left}}$ in $\mathcal{L}$ by a compact (i,ℓ,k)-labeling that actually constrains the position of L_j *less* than $\mathcal{L}_{\text{left}}$. Hence $X^\ell_j=\max\{x_j,\mathcal{X}_{i,\ell,k}+W_j\}$. Since $\ell<j$ we obtain from the induction hypothesis that $\mathcal{X}_{i,\ell,k}=\mathcal{T}[i,\ell,k]$, i.e., the values in $\theta^k_{i,j}$ are actually $X^\ell_j=\max\{x_j,\mathcal{T}[i,\ell,k]+W_j\}$.

It remains to show that a value X^ℓ_j is in $\theta^k_{i,j}$ if and only if L_ℓ is a feasible predecessors in $F(i,j,k)$. For the only-if-part, let L_ℓ be a feasible predecessor in $F(i,j,k)$. Then obviously $i\le\ell<j$. Moreover, $\mathcal{X}_{i,\ell,k}=\mathcal{T}[i,\ell,k]\le x_j$ since otherwise L_j would be pushed too far to the right for being part of an (i,j,k)-labeling. Finally, we observe that any ℓ-compact (i,j,k)-labeling induces a labeling $\mathcal{L}_{\text{right}}$ of the labels $L_{\ell+1},\dots,L_{j-1}$, in which they are restricted to lie to the right of x_ℓ, to the left of x_j and below row k. We can extend $\mathcal{L}_{\text{right}}$ to an $(\ell,j,k-1)$-labeling by placing L_ℓ at $X_\ell=x_\ell$, $Y_\ell=k-1$ and L_j at $X_j=x_j+W_j$, $Y_j=k-1$, see Fig. 4. This implies $\mathcal{T}[\ell,j,k-1]<\infty$.

For the if-part, let $i\le\ell<j$ such that $\mathcal{T}[i,\ell,k]\le x_j$ and $\mathcal{T}[\ell,j,k-1]<\infty$. We combine a compact (i,ℓ,k)-labeling $\mathcal{L}_{\text{left}}$ (exists because $\mathcal{T}[i,\ell,k]\le x_j<\infty$) and labels $L_{\ell+1},\dots,L_{j-1}$ from a compact $(\ell,j,k-1)$-labeling with the label L_j at position $X_j=x_j+W_j$ in row k as illustrated in Fig. 4. This yields a feasible (i,j,k)-labeling $\mathcal{L}$ since $\mathcal{L}_{\text{left}}$ is feasible and labels $L_{\ell+1},\dots,L_{j-1}$ lie below row k, to the right of x_ℓ, and to the left of x_j. We know that $\mathcal{T}[i,\ell,k]<x_j$ and hence L_j can be placed at x_j+W_j without overlapping L_ℓ. Furthermore both L_i and L_j are in row k and L_ℓ precedes L_j, i.e., L_ℓ is a feasible predecessor in $F(i,j,k)$. □

3.3 Weight Maximization

In this section we first show NP-hardness of MaxWeight and then present a pseudo-polynomial time algorithm showing that MaxWeight is actually weakly NP-hard.

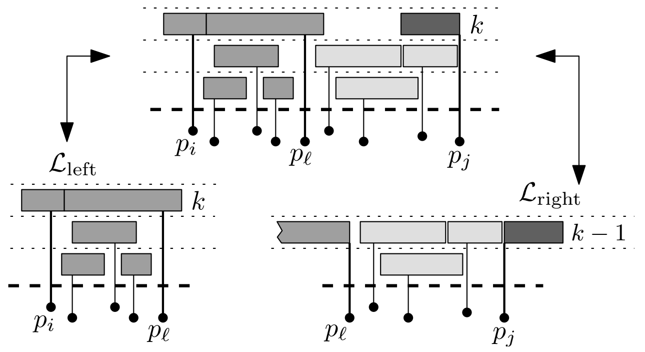


Figure 4: (De-)composition of an ℓ-compact (i,j,k)-labeling into an (i,ℓ,k)-labeling and an $(\ell,j,k-1)$-labeling.

Theorem 2. *MaxWeight is NP-hard, even for $k = 1$.*

Proof. Our proof is by reduction from the following NP-hard variant of Partition [7]: Given a set $A = \{a_1, a_2, ..., a_{2m}\}$ and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$, is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$ and A' contains exactly one of a_{2i-1}, a_{2i} for every $1 \le i \le m$? For each Partition-instance I we construct a MaxWeight-instance J such that the weight of an optimal solution to J allows us to decide whether or not I is a yes-instance of Partition. We specify the set P of points, their weights, and the widths of their labels in J as illustrated in Fig. 5. More precisely, we define a large constant $C = 1000 \sum_{a \in A} s(a)$. Next we define (from left to right) points $p_L, p_1, ..., p_{2m}, p_R$ on a horizontal line and their distances as $\overline{p_L p_1} = \overline{p_{2m} p_R} = C/2$, $\overline{p_{2i-1} p_{2i}} = (s(a_{2i-1}) + s(a_{2i}))/2$, and $\overline{p_{2i} p_{2i+1}} = C$. The corresponding labels have widths $W_L = W_R = \overline{p_L p_R}$ and $W_i = C + s(a_i)$. Furthermore, we define $w_L = w_R = W_L$ and $w_i = W_i$, i.e., the weight of a point is equal to the width of its label.

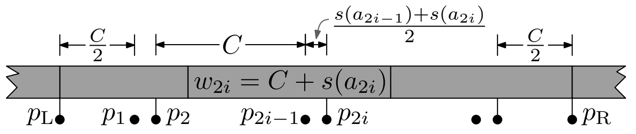


Figure 5: Reducing Partition to MaxWeight.

A solution to the partition instance I exists if and only if a feasible labeling in a single row with total weight $3\,\overline{p_L p_R}$ exists. To see why, we first assume that we are given such a labeling. Clearly, this labeling contains the label for p_L and the label for p_R. Moreover, the space between p_L and p_R is completely filled with labels for other points. Since the labels are extremely wide (compared to the distances $\overline{p_{2i-1} p_{2i}}$) exactly one of the labels L_1, L_2 has to be selected and its leader has to lie roughly in the center of the label. This again implies that exactly one of the labels L_3, L_4 has to be selected, and so on. By induction, it follows that for $i = 1, ..., m$ exactly one of the labels L_{2i-1}, L_{2i} has to be selected. Since $\overline{p_L p_R}$ is exactly half the total width of labels $L_1, ..., L_{2m}$ the labeled points correspond to a feasible partition. If, however, we are given a solution $A' \subseteq A$ to the partition instance, we can construct the labeling by selecting labels p_L, p_R plus the labels for the points corresponding to the elements in A'. It is easy to see that by applying our algorithm for SingleRow from Sect. 3.1 to the selected labels, we obtain a feasible labeling. □

Note that our construction is very similar to an NP-hardness proof for an internal 4-slider point labeling problem by Garrido et al. [8, Theorem 3]. Using the same proof it can be shown that MinRow, MaxLabels, and MaxWeight become NP-hard if labels can be placed both above and below the panorama, even if they are restricted to one row per side.

Since MaxWeight is NP-hard, we cannot hope for a polynomial-time algorithm unless $\mathcal{P} = \mathcal{NP}$. But the reduction used labels with extremely high weights. Hence, we propose a pseudo-polynomial time algorithm, i.e., an algorithm whose running time is polynomial in n, k, and the numeric value of $\sum_{i=1}^n w_i$, but exponential in the length of the encoding of the weight sum $\sum_{i=1}^n w_i$. If the label weights are small integer numbers, e.g., in the range one to four, such an algorithm can still be fast in practice.

We first extend our notation from Sect. 3.2. Recall that for a labeling $\mathcal{L}$ the set S contains all labels selected to be displayed. For a pair of indices $0 \le i \le j \le n+1$, a row index k, and a weight c we define an (i,j,k,c)-labeling as a feasible labeling of $P_{i,j} = \{p_i, \dots, p_j\}$ such that L_i and L_j are in S and placed in row k, all other labels $L_\ell \in S$ are placed in row k or below, and the total weight of the labels in S is c. Analogously, we say an (i,j,k,c)-labeling is *compact* if there is no other (i,j,k,c)-labeling where L_j has a smaller x-coordinate. Again, we call the x-coordinate of L_j in a compact (i,j,k,c)-labeling $\mathcal{L}$ the x-value of $\mathcal{L}$. Note that this definition generalizes our definition of a compact (i,j,k)-labeling, since an (i,j,k,c)-labeling is an (i,j,k)-labeling if $c = \sum_{\ell=i}^j w_\ell$.

In our algorithm for MaxWeight we add a fourth dimension to the table $\mathcal{T}$ that allows us to distinguish labelings of different weights. Let $w_{\text{total}} = \sum_{i=1}^n w_i$ be the total weight of all points in P. Then each entry $\mathcal{T}[i,j,k,c]$ stores the x-value of a compact (i,j,k,c)-labeling, where $0 \le i \le j \le n+1$, $1 \le k \le \lceil n/2 \rceil$, and $0 \le c \le w_{\text{total}}$. The maximum-weight labeling using K rows can be constructed by backtracking from $\mathcal{T}[0,n+1,K,c_{\max}]$, where $c_{\max} = \max\{c \mid \mathcal{T}[0,n+1,k,c] < \infty\}$.

As before, we compute $\mathcal{T}$ in a bottom-up fashion with respect to the topmost row k, the weight c, and the distance $j - i$. We set $\mathcal{T}[i,i,k,w_i] = x_i$ for all k and $\mathcal{T}[i,i,k,c] = \infty$ for all k and $c \ne w_i$. In all other cases we use the recurrence $\mathcal{T}[i,j,k,c] = \min \theta_{i,j}^{k,c}$, where $\theta_{i,j}^{k,c}$ for $k = 1$ is defined as the set

$$\theta_{i,j}^{1,c} = \left\{ \max\{x_j, \mathcal{T}[i,\ell,1,a] + W_j\} \;\middle|\; \begin{array}{c} i \le \ell < j, \\ \mathcal{T}[i,\ell,1,a] \le x_j, \\ a = c - w_j \end{array} \right\}$$

and for $k > 1$ as the set

$$\theta_{i,j}^{k,c} = \left\{ \max\{x_j, \mathcal{T}[i,\ell,k,a] + W_j\} \;\middle|\; \begin{array}{c} i \le \ell < j, \\ \mathcal{T}[i,\ell,k,a] \le x_j, \\ \mathcal{T}[\ell,j,k-1,b] < \infty, \\ a = c - b + w_\ell \end{array} \right\}.$$

Theorem 3. *MaxWeight can be solved by dynamic programming in $O(kn^3 w_{\text{total}}^2)$ time.*

Proof. The dynamic programming algorithm for MaxWeight is similar to Algorithm 1, but uses four instead of three nested loops to compute the $O(kn^2 w_{\text{total}})$ entries of $\mathcal{T}$. Each entry is computed as the minimum of a set $\theta_{i,j}^{k,c}$ containing $O(nw_{\text{total}})$ elements. This yields an overall running time of $O(kn^3 w_{\text{total}}^2)$.

We now show the correctness of the algorithm analogously to the proof of Theorem 1 but taking the weight constraints into account. For the case $i = j$ and arbitrary k it is easy to see that the x-value of a compact (i, i, k, c)-labeling is the leftmost possible position x_i of L_i if $c = w_i$ and ∞ otherwise.

Before we consider the general case, we extend the notion of ℓ-compact as introduced in Sect. 3.2. We call a pair (L_ℓ, a) a feasible predecessor pair of (L_j, c) if there exists an (i, j, k, c)-labeling $(i < j)$, where L_ℓ precedes L_j and the total weight of the selected labels $S \cap \{L_i, \dots, L_\ell\}$ is a. We define $F(i, j, k, c)$ as the set of all feasible predecessors pairs of (L_j, c) in an (i, j, k, c)-labeling. We then define an (i, j, k, c)-labeling $\mathcal{L}$ to be (ℓ, a)-compact if (L_ℓ, a) precedes (L_j, c) and there is no other (i, j, k, c)-labeling $\mathcal{L}'$ where (L_ℓ, a) precedes (L_j, c) and which has smaller x-value than $\mathcal{L}$. As before it is clear that every compact (i, j, k, c)-labeling $\mathcal{L}$ is also (ℓ, a)-compact for the predecessor pair (L_ℓ, a) of (L_j, c). Conversely, the (ℓ, a)-compact (i, j, k, c)-labeling with smallest x-value over all feasible predecessor pairs $(L_\ell, a) \in F(i, j, k, c)$ is compact. For every (ℓ, a)-compact (i, j, k, c)-labeling $\mathcal{L}$ the x-value of $\mathcal{L}$ is $X_j^{\ell,a} = \max\{x_j, X_\ell + W_j\}$. Then the x-value of a compact (i, j, k, c)-labeling is $\min\{X_j^{\ell,a} \mid (L_\ell, a) \in F(i, j, k, c)\}$. Our claim is that $\theta_{i,j}^{k,c} = \{X_j^{\ell,a} \mid (L_\ell, a) \in F(i, j, k, c)\}$ and thus the algorithm is correct.

Let $\mathcal{L}$ be an (ℓ, a)-compact (i, j, k, c)-labeling and let $\mathcal{L}_{\text{left}}$ be the induced (i, ℓ, k, a)-labeling. As in the proof of Theorem 1 we can assume that $\mathcal{L}_{\text{left}}$ is compact and hence by the induction hypothesis $X_j^{\ell,a} = \max\{x_j, \mathcal{T}[i, \ell, k, a] + W_j\}$, i.e., the values in $\theta_{i,j}^{k,c}$ are actually $X_j^{\ell,a}$ for some pairs (ℓ, a).

It remains to show that $X_j^{\ell,a} \in \theta_{i,j}^{k,c}$ if and only if the pair $(L_\ell, a) \in F(i, j, k, c)$. We start with the case $k = 1$. If $(L_\ell, a) \in F(i, j, 1, c)$, then obviously $i \le \ell < j$ and also $\mathcal{T}[i, \ell, 1, a] \le x_j$ since otherwise L_j cannot be in a feasible position. Moreover, for the total weight of the labeling to be c, the weight of the labels in $S \cap \{L_i, \dots, L_\ell\}$ must be $c - w_j$, since in a single row L_j is the only label to the right of L_ℓ. If, on the other hand, the three constraints for the set $\theta_{i,j}^{1,c}$ hold, we can combine a compact $(i, \ell, 1, a)$-labeling (which exists due to $\mathcal{T}[i, \ell, 1, a] < \infty$) with weight a and the label L_j with weight w_j placed in row 1 at position $X_j = x_j + W_j$ into an $(i, j, 1, c)$-labeling for $c = a + w_j$. Therefore, (L_ℓ, a) is indeed a feasible predecessor pair in $F(i, j, 1, c)$.

In the general case for $k > 1$ the argument is similar to $k = 1$. Let first $(L_\ell, a) \in F(i, j, k, c)$ be a feasible predecessor pair. Then $i \le \ell < j$ and $\mathcal{T}[i, \ell, k, a] \le x_j$ as before, but additionally any (ℓ, a)-compact labeling induces a labeling $\mathcal{L}_{\text{right}}$ of labels $S \cap \{L_{\ell+1}, \dots, L_{j-1}\}$ that is strictly below row k, to the right of x_ℓ and to the left of x_j. Furthermore, $\mathcal{L}_{\text{right}}$ has weight $c - a - w_j$. Again, we extend $\mathcal{L}_{\text{right}}$ to an $(\ell, j, k - 1, b)$-labeling by placing L_ℓ and L_j in row $k - 1$ with x-coordinates $X_\ell = x_\ell$ and $X_j = x_j + W_j$, similar to the situation depicted in Fig. 4. Note that the weight b of this labeling is $b = c - a - w_j + w_\ell + w_j$ so that all constraints put on set $\theta_{i,j}^{k,c}$ are satisfied.

Conversely, if all constraints for $\theta_{i,j}^{k,c}$ are satisfied we can compose a feasible (i, j, k, c)-labeling $\mathcal{L}$ from a compact (i, ℓ, k, a)-labeling $\mathcal{L}_{\text{left}}$ and a compact $(\ell, j, k - 1, b)$-labeling $\mathcal{L}_{\text{right}}$ as sketched in Fig. 4. The labels in $S \cap \{L_i, \dots, L_\ell\}$ are placed as in $\mathcal{L}_{\text{left}}$, the labels in $S \cap \{L_{\ell+1}, \dots, L_{j-1}\}$ as in $\mathcal{L}_{\text{right}}$ and L_j at $x_j + W_j$. The weights of the sub-labelings are chosen such that the weight of $\mathcal{L}$ correctly adds up to c. □

3.4 Label Number Maximization

In this section we present an algorithm to solve MaxLabels, i.e., to place as many labels as possible in K given rows. Note that we could solve this problem by the MaxWeight algorithm of the previous section if we set $w_i = 1$ for all $1 \le i \le n$. However, this would result in a running time of $O(kn^5)$. Here we show that we can actually do better by using an exchange argument based on the fact that all labels have the same weight.

We first introduce an adapted notation for cardinality-maximal labelings. We define a *cm-(i, j, k)-labeling* to be a feasible labeling of a subset $\hat{P} \subseteq P_{i,j} = \{p_i, \dots, p_j\}$ in rows 1 to k with both L_i and L_j placed in row k such that there is no feasible labeling of another subset $\bar{P} \subseteq P_{i,j}$ with the same properties but $|\bar{P}| > |\hat{P}|$. We say a cm-(i, j, k)-labeling $\mathcal{L}$ is *cm-compact* if there is no other cm-(i, j, k)-labeling with smaller x-value.

We will compute two three-dimensional tables $\mathcal{T}$ and $\mathcal{K}$. An entry $\mathcal{T}[i, j, k]$ for $0 \le i \le j \le n+1$ and $1 \le k \le K$ stores the x-value of a cm-compact cm-(i, j, k)-labeling. We note that $\mathcal{T}$ can no longer contain the value ∞ since for any i, j, k there is always a feasible labeling with $\hat{P} = \{p_i, p_j\}$ and their labels placed disjointly in row k. An entry $\mathcal{K}[i, j, k]$ stores the actual cardinality of a cm-(i, j, k)-labeling.

The recursive definitions of $\mathcal{T}$ and $\mathcal{K}$ are as follows:

$$\mathcal{T}[i, j, k] = \begin{cases} x_i & \text{if} \quad i = j \\ \min \theta_{i,j}^k & \text{else} \end{cases}, \tag{1}$$

with $\theta_{i,j}^k =$

$$\left\{ \max\{x_j, \mathcal{T}[i, \ell, k] + W_j\} \;\middle|\; \begin{array}{c} i \le \ell < j,\ \mathcal{T}[i, \ell, k] \le x_j, \\ \mathcal{K}[i, j, k] = \\ \mathcal{K}[i, \ell, k] + \mathcal{K}[\ell, j, k - 1] - 1 \end{array} \right\},$$

$$\mathcal{K}[i, j, k] = \begin{cases} 1 & \text{if} \quad i = j \\ \max \kappa_{i,j}^k & \text{else} \end{cases}, \tag{2}$$

with

$$\kappa_{i,j}^k = \left\{ \mathcal{K}[i, \ell, k] + \mathcal{K}[\ell, j, k - 1] - 1 \;\middle|\; \begin{array}{c} i \le \ell < j, \\ \mathcal{T}[i, \ell, k] \le x_j \end{array} \right\}.$$

The dynamic-programming algorithm for MaxLabels computes $\mathcal{K}$ and $\mathcal{T}$ in a bottom-up fashion analogously to our previous algorithms. Note that for each triple i, j, k we first compute $\mathcal{K}[i, j, k]$ and then based on that $\mathcal{T}[i, j, k]$. The final solution can be obtained by backtracking from $\mathcal{T}[0, n+1, K]$ and has $\mathcal{K}[0, n + 1, K] - 2$ selected labels.

Theorem 4. *MaxLabels can be solved by dynamic programming in $O(kn^3)$ time.*

Proof (Sketch). The running time of the dynamic-programming algorithm follows from the fact that the tables are both of size $O(kn^2)$ and computing each entry consists of finding the minimum or maximum of a set of $O(n)$ elements.

The correctness proof follows exactly the same arguments about the decomposition of a cm-(i, j, k)-labeling $\mathcal{L}$ into two labelings $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ by splitting $\mathcal{L}$ at the predecessor L_ℓ of L_j in row k (Fig. 4). The definition of $\theta_{i,j}^k$ implies that a value $X_j^\ell = \max\{x_j, \mathcal{T}[i,\ell,k] + W_j\}$ is contained in the set if and only if it leads to a cm-(i,j,k)-labeling; this is achieved by requiring $\mathcal{K}[i,j,k] = \mathcal{K}[i,\ell,k] + \mathcal{K}[\ell,j,k-1] - 1$. Note that here we need to subtract 1 because label L_ℓ is counted twice otherwise. The set $\kappa_{i,j}^k$ contains the cardinalities of all feasible ($\mathcal{T}[i,\ell,k] \leq x_j$) compositions of a cm-compact cm-(i,ℓ,k)-labeling and a cm-compact cm-$(\ell,j,k-1)$-labeling. Entry $\mathcal{K}[i,j,k]$ is then simply the maximum value in $\kappa_{i,j}^k$.

The interesting aspect for showing the correctness is the following exchange argument. Assume that there is a cm-(i,j,k)-labeling $\mathcal{L}$ with predecessor L_ℓ of L_j but $\mathcal{T}[i,\ell,k] > x_j$, i.e., the x-value of this labeling is not contained in $\theta_{i,j}^k$. Since $\mathcal{T}[i,\ell,k] > x_j$ it follows that the sub-labeling $\mathcal{L}'$ for $P_{i,\ell}$ induced by $\mathcal{L}$ is not cardinality-maximal (recall that $\mathcal{T}[i,\ell,k]$ stores the smallest x-value of all cm-(i,j,k)-labelings). So in order to have L_ℓ as the predecessor of L_j some other label left of L_ℓ in row k must be removed from the selected labels, i.e., we loose at least one label from S. But since all labels are worth the same, we can just as well remove L_ℓ itself from S and use label $L_{\ell'}$, the predecessor of L_ℓ in a cm-compact cm-(i,ℓ,k)-labeling, as L_j's predecessor. Since $L_{\ell'}$ is the predecessor of L_ℓ in that labeling we know $\mathcal{T}[i,\ell',k] \leq x_\ell < x_j$, so that the x-value $X_j^{\ell'}$ is in fact contained in $\theta_{i,j}^k$.

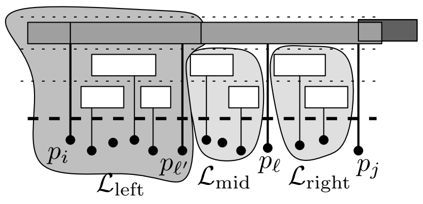

p_i
$p_{\ell'}$
$\mathcal{L}_{\text{left}}$
$\mathcal{L}_{\text{mid}}$
p_ℓ
$\mathcal{L}_{\text{right}}$
p_j

Figure 6: Construction of a labeling for Theorem 4.

It remains to argue that this labeling with predecessor $L_{\ell'}$ is at least as good as $\mathcal{L}$. By removing label L_ℓ from the cm-compact cm-(i,ℓ,k)-labeling we obtain a labeling of a subset of $P_{i,\ell-1}$ that contains exactly $\mathcal{K}[i,\ell,k]-1$ labels but still has at least as many labels as the part of $\mathcal{L}$ for points $P_{i,\ell}$. This labeling is further split into the part $\mathcal{L}_{\text{left}}$ from p_i to $p_{\ell'}$ and the part $\mathcal{L}_{\text{mid}}$ from $p_{\ell'+1}$ to $p_{\ell-1}$, see Fig. 6. Finally, there is a labeling $\mathcal{L}_{\text{right}}$ for the points $P_{\ell+1,j-1}$ that in $\mathcal{L}$ has at most $\mathcal{K}[\ell,j,k-1]-2$ labels. Now obviously by removing the leader of p_ℓ separating $\mathcal{L}_{\text{mid}}$ from $\mathcal{L}_{\text{right}}$ we obtain that $\mathcal{K}[\ell',j,k-1] \geq \mathcal{K}[\ell',\ell,k-1] + \mathcal{K}[\ell,j,k-1] - 2$. So in total we get

$$\begin{array}{rcl}
\mathcal{K}[i,j,k] & \geq & \mathcal{K}[i,\ell',k] + \mathcal{K}[\ell',j,k-1] - 1 \\
& \geq & \mathcal{K}[i,\ell',k] + \mathcal{K}[\ell',\ell,k-1] + \mathcal{K}[\ell,j,k-1] - 3 \\
& \geq & \mathcal{K}[i,\ell,k] - 1 + \mathcal{K}[\ell,j,k-1] - 1 \\
& \geq & |\mathcal{L}|,
\end{array}$$

where $|\mathcal{L}|$ denotes the number of labels in $\mathcal{L}$. This means that our algorithm indeed finds a labeling that has at least as many labels as $\mathcal{L}$. □

4. EXTENSIONS

In this section we sketch several extensions to our algorithms. With some extensions we aim at handling additional constraints, e.g., to ensure that two labels or a label and a leader of another label do not come too close. With this particular constraint the problem MinRow can become infeasible if the distance between two input points is too small. Similar problems can arise with other extensions. We do not address these problems in detail; often they can be revealed and possibly resolved in a pre-processing step.

4.1 Label spacing

Our algorithms ensure that no two labels intersect and that no label intersects a leader other than its own. It is possible, however, that two labels (or a label and a leader) get arbitrarily close to each other. To avoid this, we add the requirement that the horizontal distance between two labels in the same row (or between a label and the leader of a label in a higher row) must not be smaller than a user-defined value $\varepsilon \geq 0$. Algorithm 1 ensures this requirement if we replace the condition $\mathcal{T}[i,\ell,k] \leq x_j$ in the definition of $\theta_{i,j}^k$ with $\mathcal{T}[i,\ell,k] \leq x_j - \varepsilon$. Similarly we extend our algorithms for MaxWeight and MaxLabels.

4.2 Horizontal space constraints

In Sect. 3 we tried to compute labelings that use a small number of rows but we did not care about space consumption in the horizontal direction. If we want to ensure, e.g., that all labels are placed inside R, we have to restrict the sliding of labels. This can be done by changing the definition of the two dummy points p_0 and p_{n+1} in Sect. 3.2, i.e., we set $x_0 = x_{\min}$ and $x_{n+1} = x_{\max}$, where $x_{\min}$ ($x_{\max}$) is the smallest (largest) allowed x-coordinate for the left (right) boundary of a label. Then, for instance, a $(0, n+1, k)$-labeling with the minimum feasible value for k is an optimal solution to the horizontally constrained version of MinRow.

4.3 360° panoramas

We consider a 360°-panorama as an image curved around the inside of a cylinder, thus it does not have a left or right boundary. We can extend our algorithm for MinRow to handle such images. Our extension is based on the following observation: if there is a solution to MinRow with k rows, there is also a solution with k rows that contains one of the labels L_i at position $X_i = x_i$, $Y_i = k$.

Our idea is to extend table $\mathcal{T}$, which so far contained values $\mathcal{T}[i,j,k]$ only for $i \leq j$. The new version of the algorithm computes values $\mathcal{T}[i,j,k]$ *for all* $1 \leq i,j \leq n$ and $1 \leq k \leq k^*$, where k^* is the smallest feasible number of rows. If $j < i$, a labeling corresponding to $\mathcal{T}[i,j,k]$ simply contains labels $L_{i+1}, ..., L_n$ and $L_1, ..., L_j$. We compute $\mathcal{T}[i,j,k]$ as before, ignoring the fact that in a 360°-panorama L_i and L_j may intersect in the interval bounded to the left by x_j and to the right by x_i. Nevertheless, it is easy to see that the values in $\mathcal{T}$ allow us to decide for $i = 1, ..., n$ and $k = 1, ..., \lceil n/2 \rceil$ whether there is a feasible labeling in k rows such that $X_i = x_i$, $Y_i = k$, and all labels fit into the interval between $x_i - 360°$ and x_i. With this change, Algorithm 1 still needs $O(k^* n^3)$ time.

4.4 Elite labels

For the most important sites, e.g., the Willis Tower in the panorama of Chicago (see Figures 1 and 7), we may want to guarantee that the solution contains the corresponding labels. We term such a label an *elite label*. Now we can ask for a feasible labeling that contains all elite labels plus as many other labels as possible in a given number k of rows. Assuming that such a solution exists, we can apply our algorithm for MaxWeight. We simply have to set the weight of each elite label to a large enough number (e.g.,

$n+1$) and the weight of each non-elite label to one. With this approach we have $w_{\text{total}} \in O(n^2)$, thus the algorithm requires $O(kn^7)$ time. With the following simple modifications of recurrences (2) and (1), however, we can reuse our $O(kn^3)$-time algorithm for MaxLabels.

Each $(i, j, 1)$-labeling with $j > i$ contains a label L_ℓ preceding L_j. We have to avoid that we omit an elite label between L_i and L_j. To do so, we exchange the condition $i \leq \ell < j$ in the definitions of $\kappa^1_{i,j}$ and $\theta^1_{i,j}$ with $e \leq \ell < j$, where L_e is the last elite label among $L_{i+1}, \ldots, L_{j-1}$. Obviously, we do not lose any feasible solution with this. Moreover, the modification suffices since our algorithm always constructs a solution based on feasible one-row solutions for smaller (sub-)instances. Similarly we can extend our algorithm for MaxWeight to find a labeling in k rows containing all elite labels plus other labels of maximum total weight.

4.5 Optimizing label positions

A secondary criterion of æsthetically pleasing panorama labelings is that the labels are as close as possible to being centered above their respective sites. So far we did not consider this criterion in our problem definitions. Moreover, the algorithms described in Sect. 3 try to place all labels at their leftmost feasible positions. We can remedy this unwanted side effect by subsequently fine-tuning the horizontal label positions. We traverse each row k of a given feasible labeling from right to left. Let S_k be the set of labels in row k. During the traversal, we slide a label to the right if it decreases the overall cost $H = \sum_{L_i \in S_k} |\Delta_i|$, where Δ_i is the horizontal distance between the center of label L_i and x_i. A right shift of a label triggers a right shift of another label if this reduces the overall cost and the labeling remains feasible. It can be shown that for a given row our positioning algorithm minimizes H in $O(n^2)$ time, so that the total post-processing time is covered by the running-time of the labeling algorithm.

5. EXPERIMENTAL RESULTS

In this section we evaluate our algorithms, which we implemented in C++ using GTK+ and Cairo for the visual output. We tested the algorithms both for real-world instances as well as randomly generated instances. For a discussion of a case study with the Chicago skyline see Fig. 7.

We executed the experiments on a single core of an AMD Opteron 2218 processor that is clocked at 2.6 GHz and has 2×1 MiB of L2 cache. The machine is running Linux 2.6.34.8 and has 16 GiB of RAM. We compiled our C++ implementation with GCC 4.5, using optimization level 3.

For each set of test parameters we generated 1,000 test instances. All input points have integer x-coordinates between 0 and 1280. These coordinates were chosen uniformly at random. The label widths were randomly chosen over a Gaussian distribution with mean 115 and standard deviation $\sigma = 20$. These values stem from real-world experience.

Row Number Minimization.

The first set of our experiments focuses on our implementation of Algorithm 1. Table 1 reports the minimum, average, and maximum running time for labeling varying numbers of labels. We observe that even for instances where the points lie unreasonably dense (e.g., instances with 150 labels where the average distance between points is roughly 8 pixels) our algorithm solves all instances in less than 0.5s. Interestingly, the vast majority of instances with 75 or more labels is formed by worst-case instances for MinRow, i.e., they require $\lceil n/2 \rceil$ rows. For reasonable real-world instances we can expect that our algorithm generates an optimal solution near-instantaneously.

#labels	10	25	50	75	100	125	150
min. [ms]	<1	<1	3	14	52	171	460
avg. [ms]	<1	<1	4	24	84	222	477
max. [ms]	<1	1	5	26	87	228	492

Table 1: Evaluation of our implementation of Row Number Miminization instances with 10–150 labels.

Label Number Maximization.

In the second set of experiments we investigate the running times of our algorithm for MaxLabels. We believe that in most practical application no more than four rows of labels are used and hence ran the first part of experiments for $K = 4$. We observe from the data in Table 2 that our algorithm performs again very well, even for large instances. In this case it even outperforms the previous MinRow algorithm. This is mainly due to the fact that many instances are worst-case instances for the MinRow algorithm.

#labels	10	25	50	75	100	125	150
min. [ms]	<1	<1	1	4	10	21	39
avg. [ms]	<1	<1	1	5	11	22	41
max. [ms]	<1	<1	1	5	17	23	46

Table 2: Label number maximization for 4 rows.

Our algorithm is also able to quickly generate solutions for more unrealistic instances where we allow the labels to be placed in at most 50 rows, see Table 3. We observe that even in the worst-case scenario, i.e., placing 150 labels in at most 50 rows, the execution time is less than 900ms.

#labels	10	25	50	75	100	125	150
min. [ms]	<1	<1	7	38	142	337	789
avg. [ms]	<1	<1	8	40	155	345	810
max. [ms]	<1	<1	9	45	163	354	895

Table 3: Label number maximization for 50 rows.

Weight Maximization.

Finally, we evaluate our MaxWeight algorithm. For this we experiment with two different types of weight distributions. In some use cases it might be useful to define a ranking of the label importance. Then, for a set of n labels, we assign each label a distinct integer weight between 1 and n. In all generated instances the weights were distributed uniformly at random. A different way of determining label weights is to group labels into classes of equal importance. Generally, we can expect that there is only a limited number of such classes. In the experiment we defined four classes and the labels were assigned to the classes uniformly at random. Labels in class i have weight 2^i.

We report the running times of our MaxWeight algorithm for both types of weight distributions in Tables 4 and 5, respectively. These results were again generated using $K = 4$ rows. Note that the measured execution times in these tables are reported in seconds and not milliseconds as before.

#labels	10	25	30	40	50	75
min. [s]	<0.01	0.16	1.62	10.3	50.47	722.56
avg. [s]	<0.01	0.17	1.81	12.1	54.76	778.2
max. [s]	<0.01	0.19	2.02	13.6	61.84	866.09

Table 4: Weight maximization for ranked labels.

#labels	10	25	50	75	100
min. [s]	<0.01	0.03	1.02	28.27	112.98
avg. [s]	<0.01	0.07	1.34	32.96	127.91
max. [s]	<0.01	0.09	1.73	38.6	137.83

Table 5: Weight maximization for labels in four importance classes and weights $\{1, 2, 4, 8\}$.

Since the algorithm has a pseudo-polynomial running time the higher execution times compared to our other algorithms were expected. Although the results reported in both tables confirm this expectation, we observe that for small and medium numbers of rows and labels, the algorithm still runs within an acceptable time frame. However, if we raise the number of available rows substantially or increase the total number of labels, the execution time grows quickly and may become unacceptable.

6. CONCLUSION

We have presented algorithms for label placement using a new boundary label model that allows multiple rows of sliding unit-height rectangular labels. In this model, each label is connected with its associated point by a vertical line segment. We have presented an $O(k^*n^3)$-time algorithm for the basic problem *MinRow*, which minimizes the number of rows needed to place all labels. If the labeling is restricted by the input to k rows, however, we cannot generally place all labels. Therefore, we have investigated the problem *MaxLabels*, which aims at maximizing the total number of labels in k rows, and the problem *MaxWeight*, which aims at maximizing the total weight of labels in k rows. While MaxLabels can be solved in $O(kn^3)$ time, MaxWeight turned out to be weakly NP-hard, yet allowed for an $O(kn^3w_{\text{total}}^2)$-time algorithm, where w_{total} is the total weight of all input labels.

According to our experiments, the algorithms for MinRow and MaxLabels are very fast for instances typically arising in practice, i.e., they solve instances with up to 150 labels in less than one second. The algorithm for MaxWeight is fast if the weights are not too large. For example, with integer weights between 1 and 8, we can solve instances with 50 labels in less than two seconds. We think that our setting is realistic, since labeled images already with more than 50 labels quickly appear visually cluttered and more than 150 labels seems unrealistic in most cases. Similarly, if sites are assigned importance levels, there are usually few of them (e.g., main landmarks, distinctive buildings, public buildings, other). We conclude that our algorithms can quickly produce visually pleasing labelings of real-world panorama images. An interesting open question is how to handle labels of different heights that would take up more than one row or how to deal with area sites instead of point sites.

Acknowledgments. This work was started at Schloss Dagstuhl during Seminar 10461 "Schematization in Cartography, Visualization, and Computational Geometry" in November 2010. Andreas Gemsa and Martin Nöllenburg received financial support by the *Concept for the Future* of KIT within the framework of the German Excellence Initiative.

7. REFERENCES

[1] M. A. Bekos, M. Kaufmann, M. Nöllenburg, and A. Symvonis. Boundary labeling with octilinear leaders. *Algorithmica*, 57(3):436–461, 2010.

[2] M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Multi-stack boundary labeling problems. In S. Arun-Kumar and N. Garg, editors, *Proc. 26th Conf. Found. Softw. Technol. and Theor. Comput. Sci. (FSTTCS'06)*, volume 4337 of *Lecture Notes Comput. Sci.*, pages 81–92. Springer-Verlag, 2006.

[3] M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Area-feature boundary labeling. *Comp. J.*, 53(6):827–841, 2010.

[4] M. A. Bekos, M. Kaufmann, A. Symvonis, and A. Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Comp. Geom. – Theor. Appl.*, 36(3):215–236, 2007.

[5] M. Benkert, H. Haverkort, M. Kroll, and M. Nöllenburg. Algorithms for multi-criteria boundary labeling. *J. Graph Algorithms Appl.*, 13(3):289–317, 2009.

[6] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annuual ACM Sympos. on Computational Geometry (SoCG'91)*, pages 281–288, 1991.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[8] M. Garrido, C. Iturriaga, A. Márquez, J. R. Portillo, P. Reyes, and A. Wolff. Labeling subway lines. In *Proc. 12th Internat. Symp. Algorithms and Computation (ISAAC'01)*, volume 2223 of *Lecture Notes Comput. Sci.*, pages 649–659. Springer, Berlin, Germany, 2001.

[9] H.-C. Y. Hao-Jen Kao, Chun-Cheng Lin. Many-to-one boundary labeling. In *Proc. Asia-Pacific Sympos. on Visualisation (APVIS'07)*, pages 65–72. IEEE, 2007.

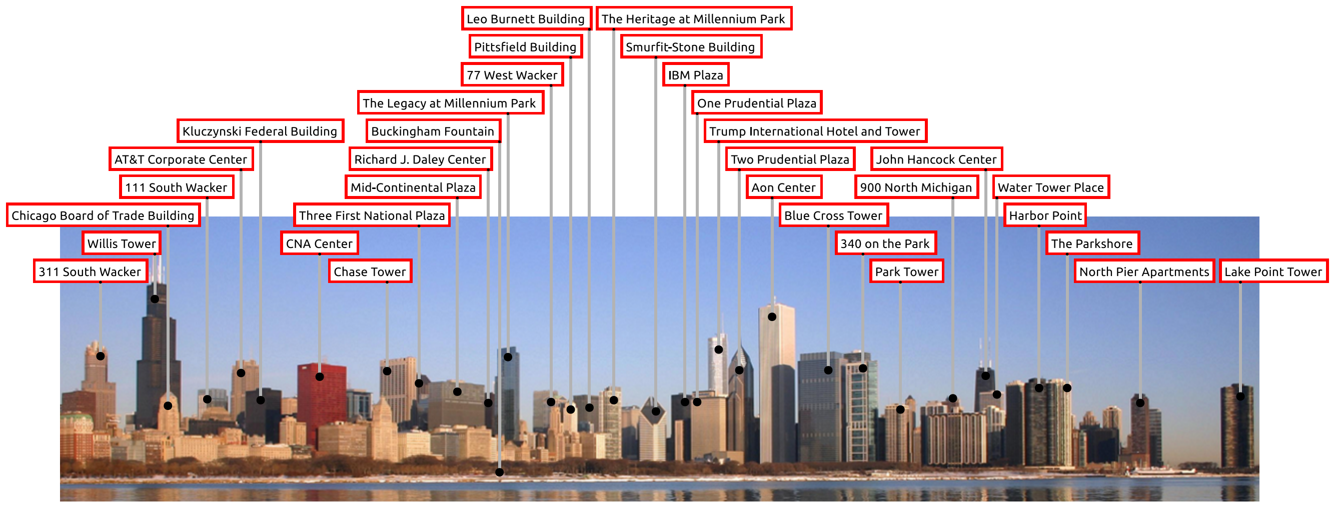
[10] M. Kaufmann. On map labeling with leaders. In S. Albers, H. Alt, and S. Näher, editors, *Festschrift Mehlhorn*, volume 5760 of *Lecture Notes Comput. Sci.*, pages 290–304. Springer-Verlag, 2009.

[11] C.-C. Lin. Crossing-free many-to-one boundary labeling with hyperleaders. In *Proc. IEEE Pacific Visualisation Symposium (PacificVis'10)*, pages 185–192, 2010.

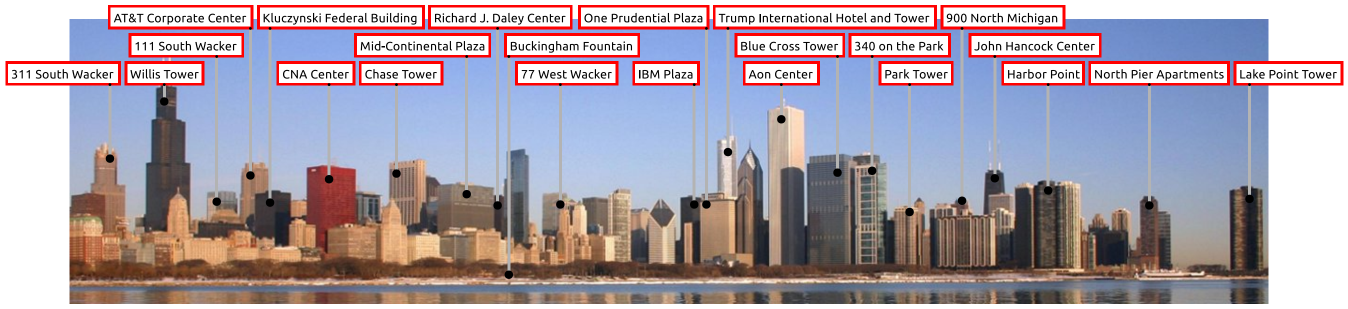
[12] M. Nöllenburg, V. Polishchuk, and M. Sysikaski. Dynamic one-sided boundary labeling. In *Proc. 18th ACM SIGSPATIAL Internat. Conf. Advances Geograph. Inform. Syst. (ACM GIS 2010)*, pages 310–319, 2010.

[13] S.-H. Poon, C.-S. Shin, T. Strijk, T. Uno, and A. Wolff. Labeling points with weights. *Algorithmica*, 38(2):341–362, 2003.

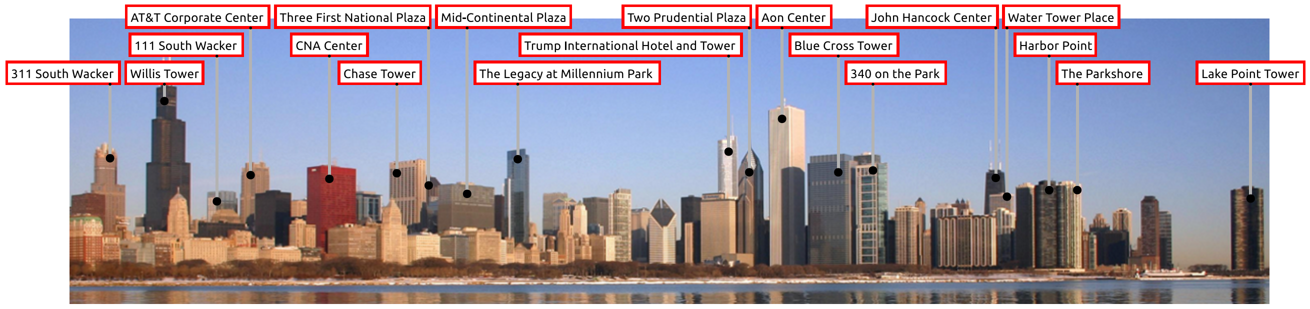
[14] A. Wolff and T. Strijk. The Map-Labeling Bibliography, 1996.



(a) Solution of a MinRow instance that requires ten rows. We observe that the solution is æsthetically not appealing. In the center of the panorama the arrangement of labels is reminiscent of the worst-case scenario illustrated in Fig. 2.



(b) Solution of a MaxLabels instance in three rows. Due to restricting the number of label rows, the result is much more pleasing than the MinRow solution (a). Of all 33 labels 23 are displayed. This indicates that only a few labels of densely placed points are responsible for the visually unpleasing MinRow result.



(c) Solution of a MaxWeight instance in three rows. We divided the buildings into four equal-size classes based on their height, i.e., the $\lceil n/4 \rceil$ tallest buildings have weight 8, and the other classes have weights 4, 2, and 1 accordingly. This yields the maximum total weight of all labels $w_{\text{total}} = 128$. Of the 33 labels 19 labels are displayed and they have a total weight of 108. Although the two labelings in 7(b) and 7(c) look similar at first sight, note that for instance the MaxWeight solution contains a label for "The Legacy at Millennium Park", the fourth tallest building in the panorama, while the MaxLabels solution misses that label.

Figure 7: A case study with the Chicago skyline using three of our algorithms. The input data for all three Figures is the same. It consists a total of 33 labels. On a laptop clocked at 2.4 Ghz it took roughly 1ms to compute the panorama labelings in Figures (a), (b) and about 160ms to compute the labeling shown in (c). Photography: ©J. Crocker (`http://en.wikipedia.org/w/index.php?title=File:2010-02-19_16500x2000_chicago_skyline_panorama.jpg`).