ELSEVIER

Computational Geometry 11 (1998) 209–218

Computational Geometry
Theory and Applications

Label placement by maximum independent set in rectangles

Pankaj K. Agarwal [a,*,1], Marc van Kreveld [b,2], Subhash Suri [c,3]

[a] *Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA*
[b] *Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands*
[c] *Department of Computer Science, Washington University, St. Louis, MO 63130, USA*

Communicated by E. Welzl; accepted 31 August 1998

Abstract

Motivated by the problem of labeling maps, we investigate the problem of computing a large non-intersecting subset in a set of n rectangles in the plane. Our results are as follows. In $O(n \log n)$ time, we can find an $O(\log n)$-factor approximation of the maximum subset in a set of n arbitrary axis-parallel rectangles in the plane. If all rectangles have unit height, we can find a 2-approximation in $O(n \log n)$ time. Extending this result, we obtain a $(1 + 1/k)$-approximation in time $O(n \log n + n^{2k-1})$ time, for any integer $k \geqslant 1$.

1. Introduction

Automated label placement is an important problem in geographic information systems (GIS), and has received considerable attention in recent years (for instance, see [6,9]). The label-placement problem includes positioning labels for area, line and point features. The primary focus within the computational geometry community has been on labeling point features [5,7,15,16]. A basic requirement in the label-placement problem is that the labels are pairwise disjoint. Subject to this basic constraint, the most common optimization criteria are the number of features labeled and the size of the labels. Other variations include the choice of the shapes of the labels and the legal placements allowed for each point. Unfortunately, even in simple settings, the problem turns out to be NP-hard [3,7].

In this paper we assume that each label is an orthogonal rectangle of fixed size and we want to place as many labels as possible. More precisely, let S be a set of n points in the plane. For each point $p_i \in S$,

* Corresponding author. E-mail: pankaj@cs.duke.edu.

[1] Research partially supported by National Science Foundation Grant CCR-93-01259, by an Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by an NYI award, and by matching funds from Xerox Corporation, and by a grant from the U.S.–Israeli Binational Science Foundation.

[2] Research partially supported by the ESPRIT IV LTR Project No. 21957 (CGAL), and by the Dutch Organization for Scientific Research (NWO).

[3] Research partially supported by NSF Grant CCR-9501494.

PII: S0925-7721(98)00028-5

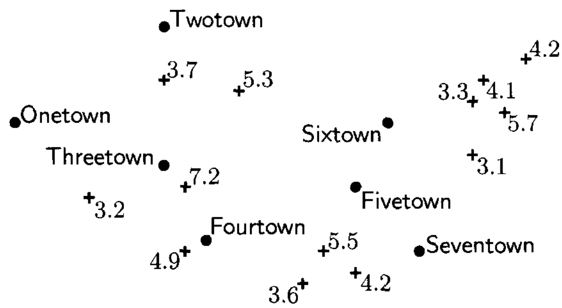

Fig. 1. Point labels that are names of towns, mixed with epicenters of earthquakes labeled with their magnitude.

we have a label r_i, and a set π_i of marked points on the boundary of r_i. Typical choices of π_i include the endpoints of the left edge of r_i, the four vertices of r_i, or the four vertices and middle points of the four edges of r_i. A valid placement of r_i is a translated copy $r_i + (p_i - x_i)$ of r_i for some $x_i \in \pi_i$, that is, r_i is placed so that one of the marked positions on the boundary of r_i coincides with the point p_i. A *feasible configuration* is a family of pairs $\{(p_{i_1}, x_{i_1}), \ldots, (p_{i_k}, x_{i_k})\}$, where all the i_j are different and $x_{i_j} \in \pi_{i_j}$, so that the rectangles in $\{r_{i_1} + (p_{i_1} - x_{i_1}), \ldots, r_{i_k} + (p_{i_k} - x_{i_k})\}$ are pairwise disjoint. The *label-placement* problem is to find a largest feasible configuration.

In practice the labels are subject to additional constraints, which help in simplifying and improving the algorithms. Restricting the shape of the labels to be same size squares is one such approach [7,15, 16], because in many technical maps all labels have the same size. Think of mapping measurements at sample points in a terrain, or maps showing magnitudes of earthquakes at points that are the epicenters. Another interesting case is when all labels have the same height but arbitrary width. This situation arises, for example, if we want to label points with names on a map and all labels are in the same font size, or when different types of point labels occur on a map. In this paper we consider the second case.

We will study the case when π_i has a constant number of positions on the boundary of r_i. The rectangles are closed; they include the boundary. Let $R_i = \{r_i + (p_i - x_j) \mid x_j \in \pi_i\}$ and set $R = \bigcup_{i=1}^{n} R_i$. The label-placement problem is the same as computing a largest subset of pairwise disjoint rectangles in R. Since all rectangles in R_i have a common intersection point p_i, at most one rectangle can be chosen from each R_i. Consider the *intersection graph* $G(R)$ of R: the nodes of $G(R)$ are the rectangles of R and there is an edge between two nodes if the corresponding rectangles intersect. A subset of pairwise disjoint rectangles in R corresponds to an independent set in $G(R)$. We want to compute a maximum independent set of $G(R)$. Abusing the terminology slightly, we will say that we want to compute a maximum independent set of R. Computing an independent set even of unit squares is known to be NP-hard [8,12]. This suggests that one should aim for approximation algorithms. We call an algorithm an *ε-approximation algorithm*, for $\varepsilon > 1$, if it returns an independent set of size at least γ/ε, where γ is the size of a maximum independent set of R.

Although it is known that no polynomial-time $\Omega(n^{1/4})$-approximation algorithm exists for maximum independent sets in arbitrary graphs [1], no such lower bound is known for intersection graphs of rectangles. In this paper we present an $O(n \log n)$-time $(\log n)$-approximation algorithm for rectangles. [1] For the case that all rectangles in R have the same height, we describe an $(1 + 1/k)$-approximation

[1] All logarithms in this paper are base 2.

algorithm whose running time is $O(n \log n + n^{2k-1})$, for any $k \geqslant 1$. This is an important case, since it models the label-placement problem when all labels have the same font size. It is an open problem whether a c-approximation algorithm exists for arbitrary rectangles, for any positive constant c.

The paper is organized as follows. Section 2 summarizes the previous work on the label-placement problem. In Section 3 we describe the approximation algorithm for arbitrary orthogonal rectangles. Section 4 describes our approximation algorithm for unit-height rectangles, which is based on dynamic programming.

2. Previous research

There has been much work on label placement in the cartography community; see, e.g., [6,9] and the references therein for a sample of results. Algorithms researchers have also studied labeling maps. Formann and Wagner [7] considered the label placement for point features in the plane using *square* labels. Specifically, an axis-aligned square label is placed for each point such that the point coincides with one of the vertices of its labeling square. They used the *size* of the square label as the optimization criterion, subject to the condition that all points must receive a label. The square represents the text or measurement to be placed at the point. Their optimization is motivated by the maximum font size: since the problem allows scaling in the x-direction, it is the same as rectangular label placement for equal-size labels.

Given a point p, there are four positions for placing a square label so that the point coincides with one of the corners of the label. If all four positions of labels are allowed, then the problem of determining whether labels of given size can be placed is NP-complete. Formann and Wagner give an $O(n \log n)$ time algorithm that guarantees a label size at least half the optimum [7]. They also show that no better approximation is possible unless P = NP. Formann and Wagner's approach is to grow all four possible labels around the points, removing candidate placements when they conflict with other growing labels. Whether the remaining labels allow a placement is done by solving 2-SAT problems. Kučera et al. [13] studied the same problem, but developed an exact, super-polynomial algorithm that can be applied for sets with up to roughly 100 points.

Wagner and Wolff [15,16] have noted that, in practice, the approach of Formann and Wagner hardly ever results in square sizes significantly greater than half the optimum. They also study several variations and their implementation and find ways to improve on the size of the squares in practice.

Doddi et al. [5] allow more general shapes of labels, e.g., circles, nonoriented rectangles, ellipses, and present approximation algorithms in each case. Like Formann and Wagner, they also approximate the size of labels. See also [11,14].

Christensen et al. [3] provide a comparison of several approaches to place as many labels as possible on a map. They consider point labels, line labels, and area labels. A further comparison can be found in [2].

3. Arbitrary size rectangular labels

We describe a simple, divide-and-conquer algorithm for computing a large independent set in a set R of n orthogonal rectangles in the plane. We sort the horizontal edges of R by their y-coordinates and their vertical edges by their x-coordinates; this step takes $O(n \log n)$ time. This sorting is done only once in the beginning. If $n \leqslant 2$, we compute the maximum independent set in $O(1)$ time. Otherwise, we do the following.

1. Let x_{med} be the median x-coordinate among the abscissae of R.
2. Partition the rectangles of R into three groups: R_1, R_2 and R_{12}, where R_{12} contains rectangles intersecting the line ℓ: $x = x_{\text{med}}$, and R_1 and R_2 contains the rectangles lying to the left and right, respectively, of the line.
3. Compute I_{12}, the (real) maximum independent set of R_{12}. Recursively compute I_1 and I_2, the approximate maximum independent sets in R_1 and R_2, respectively.
4. If $|I_{12}| \geqslant |I_1| + |I_2|$, return I_{12}, otherwise return $I_1 \cup I_2$.

The first observation is that the rectangles in R_1 are disjoint from the rectangles in R_2. Consequently, $I_1 \cup I_2$ is an independent set. The second observation is that since all rectangles in R_{12} intersect the line ℓ, it suffices to compute a largest nonoverlapping subset of intervals in the set $J = \{r \cap \ell \mid r \in R_{12}\}$, in order to compute I_{12}. This one-dimensional problem can be solved optimally by the following greedy strategy in $O(n \log n)$ time. Sort the intervals in the ascending order of their bottom endpoints. Add the topmost interval l, with highest bottom endpoint, to the independent set; delete all intervals intersecting l; and repeat until no intervals remain. Recall that the horizontal edges of rectangles in R are sorted by their y-coordinates, so we can sort the intervals in J by their bottom endpoints in linear time. Since $|R_1| \leqslant |n|/2$ and $|R_2| \leqslant |n|/2$, the overall running time of the algorithm is $O(n \log n)$.

Next, we prove by induction that our algorithm computes an independent set of size at least $\gamma / \max(1, \log n)$, where γ is the largest independent set. For $n \leqslant 2$, we compute a largest independent set, so the claim is obviously true for $n \leqslant 2$. Suppose it is true for all $m < n$. Let I^* be a maximum independent set of R. Similarly, let I_1^*, I_2^* and I_{12}^* be the maximum independent sets of R_1, R_2 and R_{12}, respectively. Since the algorithm computes a maximum independent set I_{12} of R_{12}, we have $|I_{12}| = |I_{12}^*| \geqslant |I^* \cap R_{12}|$. By the induction hypothesis,

$$|I_1| \geqslant \frac{|I_1^*|}{\log(n/2)} \geqslant \frac{|I^* \cap R_1|}{\log n - 1} \quad \text{and similarly,} \quad |I_2| \geqslant \frac{|I^* \cap R_2|}{\log n - 1}.$$

Therefore,

$$\begin{aligned} |I| = \max\{|I_{12}|, |I_1| + |I_2|\} &\geqslant \max\left\{|I^* \cap R_{12}|, \frac{|I^* \cap R_1| + |I^* \cap R_2|}{\log n - 1}\right\} \\ &\geqslant \max\left\{|I^* \cap R_{12}|, \frac{|I^*| - |I^* \cap R_{12}|}{\log n - 1}\right\}. \end{aligned}$$

If $|I^* \cap R_{12}| \geqslant |I^*| / \log n$, the induction step is proved. Otherwise,

$$\frac{|I^*| - |I^* \cap R_{12}|}{\log n - 1} \geqslant \frac{|I^*| - |I^*|/\log n}{\log n - 1} = \frac{|I^*|}{\log n},$$

and the induction step is proved as well. Hence, we obtain the following result.

Theorem 1. *Let R be a set of n axis-parallel rectangles in the plane. An independent set of R of size at least $\gamma / \log n$ can be computed in time $O(n \log n)$, where γ is the size of a maximum independent set in R.*

4. Approximation scheme for unit-height rectangles

In this section we develop a polynomial-time approximation algorithm for computing an independent set of rectangles of fixed height, but of arbitrary width. As discussed earlier, our class is clearly more general than unit squares, and this added generality is important for labeling maps. We assume without loss of generality that all rectangles have unit height. We first develop a 2-approximation algorithm that takes $O(n \log n)$ time. Then, using dynamic programming, we obtain a $(1 + 1/k)$-approximation algorithm whose running time is $O(n \log n + n^{2k-1})$ time, for any $k \geqslant 1$.

4.1. A 2-approximation algorithm

Consider a set R of n unit-height rectangles in the plane. We draw a set of horizontal lines, $\ell_1, \ell_2, \ldots, \ell_m$, where $m \leqslant n$, so that the following three conditions hold:
(1) the separation between two lines is strictly more than one,
(2) each line intersects at least one rectangle, and
(3) each rectangle is intersected by some line.
Note that minimum separation condition implies that a rectangle cannot be intersected by more than one line. The lines can be drawn from top to bottom using an incremental approach. These lines partition the set R into subsets $R_1, R_2, \ldots, R_m$, where R_i is the set of rectangles in R that intersect line ℓ_i.

We compute a maximum independent set M_i for each R_i, which takes $O(|R_i| \log |R_i|)$ time, using the one-dimensional greedy algorithm. Since the line ℓ_i does not intersect any rectangle of $R \setminus R_i$, the rectangles in M_i do not intersect any rectangle of M_j except for $j = i - 1$ or $j = i + 1$. Consider the two independent sets $\{M_1 \cup M_3 \cup \cdots \cup M_{2\lceil m/2\rceil-1}\}$ and $\{M_2 \cup M_4 \cup \cdots \cup M_{2\lfloor m/2\rfloor}\}$. Clearly, the larger of these two must have size at least $\gamma/2$, and thus we have a 2-approximation algorithm. The running time of the algorithm is $O(n \log n)$, since finding the lines ℓ_i and forming the corresponding partition can be done in a single pass through the rectangles after sorting them by their y-coordinates.

Theorem 2. *Let R be a set of n unit height axis-parallel rectangles in the plane. In $O(n \log n)$ time, we can compute an independent set of size at least $\gamma/2$, where γ is the size of a maximum independent set of R.*

4.2. A $(1 + 1/k)$-approximation algorithm

We will improve the approximation factor to $(1 + 1/k)$, for any $k \geqslant 1$, by combining dynamic programming with the shifting technique of Hochbaum and Maass [10]. The basic idea is to partition the rectangles by horizontal lines $\ell_1, \ell_2, \ldots, \ell_m$ as before, but then use dynamic programming to *optimally* solve the subproblem for each subset of rectangles intersected by k consecutive lines. Suppose the lines are labeled $\ell_1, \ell_2, \ldots, \ell_m$ from top to bottom, and R_i is the set of rectangles intersecting the line ℓ_i. We set $R_i = \emptyset$ for $i > m$. Define $R_i^k = R_i \cup R_{i+1} \cup \cdots \cup R_{i+k-1}$, that is, R_i^k is the set of rectangles intersecting any line in the set $\{\ell_i, \ell_{i+1}, \ldots, \ell_{i+k-1}\}$. We will refer to the R_i^k's as *subgroups*.

We now define $k + 1$ groups $G_1, \ldots, G_{k+1}$ (see Fig. 2), where

$$G_j = R_1^{j-1} \cup \bigcup_{i \geqslant 0} R^k_{i(k+1)+j} = R \Big\backslash \bigcup_{i \geqslant 0} R_{i(k+1)+j}.$$

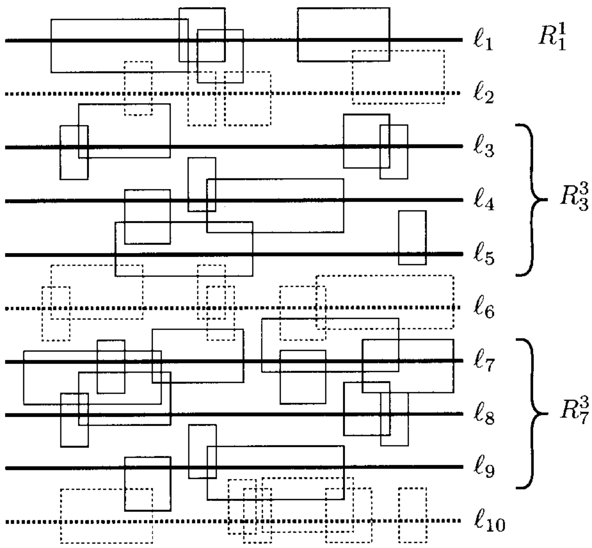

Fig. 2. The group G_2 for $k = 3$.

That is, for $1 \leqslant j \leqslant k+1$ the group G_j is obtained from R by deleting rectangles intersected by every $(k+1)$st line, starting with the ℓ_jth line.

We make two key observations about these groups of rectangles. First, consider two consecutive subgroups within any group, such as R_1^k and R_{k+2}^k in G_1. No rectangle of R_1^k intersects a rectangle in R_{k+2}^k; the line ℓ_{k+1} separates these subgroups. By extension, this means that for any group G_j, the rectangles in a subgroup of G_j are disjoint from the rectangles of any other subgroup of G_j. Thus, if we combine the independent sets for all the subgroups, we get an independent set of the whole group G_j. Second, since a group is formed by deleting all rectangles that intersect every $(k+1)$st line, the union of all rectangles in $R \setminus G_j$ is intersected by at most $\lceil m/(k+1) \rceil$ lines. If we compute a maximum independent set for each G_j and choose the largest one, we can miss at most $\gamma/(k+1)$ rectangles by the pigeon hole principle. Hence we get an $(1 + 1/k)$ factor approximation. This is exactly the shifting idea of Hochbaum and Maass [10], and this is precisely what we will do as well.

We give a dynamic programming solution for computing a maximum independent set $M(R_j^k)$ for any subgroup R_j^k, that is, a set of rectangles intersected by k consecutive lines in $\ell_1, \ell_2, \ldots, \ell_m$. After computing $M(R_j^k)$ for every $j \geqslant 1$, the rest of the algorithm is straightforward.

For ease of exposition, we describe the algorithm for the case $k = 2$, but all the ideas generalize readily. Without loss of generality, let us consider the problem of computing a maximum independent set for $R_1^2 = R_1 \cup R_2$, that is, the rectangles intersecting ℓ_1 or ℓ_2. Let $X = (x_1, x_2, \ldots, x_g)$ denote the sequence of distinct abscissae of the vertical edges of $R_1 \cup R_2$, sorted in increasing order (left to right). Note that only the ordinates of the bottom edges of rectangles in R_1 and of the top edges of rectangles in R_2 are relevant. Let $Y = (y_1, y_2, \ldots, y_h)$ denote the sequence of distinct ordinates of bottom edges from R_1 and of top edges from R_2, sorted in the increasing order (bottom to top). Add to X the value x_0 where $x_0 < x_1$. Add to Y the value y_0, which is the ordinate of the line ℓ_2 minus 1, and the value y_{h+1}, which is the ordinate of the line ℓ_1 plus 1. Note that $y_0 < y_1$ and $y_h < y_{h+1}$.

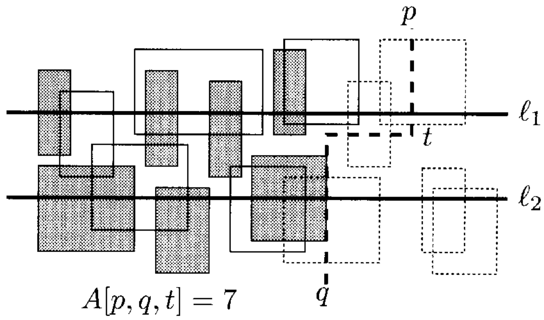

Fig. 3. Polygonal line defined by p, q, t and its relation to the table entry $A[p, q, t]$.

With each triple $\tau = (p, q, t)$, where $0 \leqslant p, q \leqslant g$ and $0 \leqslant t \leqslant h+1$, we associate a polygonal line λ_τ defined as follows: if $p = q$, then λ_τ is the vertical line $x = p$; otherwise λ_τ consists of a vertical ray emanating from the point (x_p, y_t) in the $(+y)$-direction, the horizontal segment connecting (x_p, y_t) to (x_q, y_t), and another vertical ray emanating from the point (x_q, y_t) in the $(-y)$-direction, see Fig. 3. Let $R_\tau \subseteq R$ denote the set of rectangles whose interiors lie to the left of the polyline λ_τ. Let M_τ denote a maximum independent set of R_τ, and let $A_\tau = |M_\tau|$. We now describe how we compute A_τ for all triples $\tau = (p, q, t)$. We will construct a three-dimensional table A, in which $A[p, q, t]$ will store the value of $A_{(p,q,t)}$.

We consider the case when $p > q$; the case $p < q$ is symmetric. If $p = q$, the third index t plays no role. We can fill out the entry $A[p, p, ..]$ as the case where $p > q$ and the third index is $h + 1$. So we need only consider the case $p > q$.

If $p > q$ and no rectangle in $R_\tau \cap R_1$ has its right edge at $x = x_p$, then $R_\tau = R_{(p-1,q,t)}$; therefore $A[p, q, t] = A[p-1, q, t]$. Otherwise, let $r \in R_1$ be the rectangle whose right edge is at $x = x_p$. (Let us assume that there is only one such rectangle; we will discuss later how to handle the case when the right edges of many rectangles lies on the line $x = x_p$.) Suppose the left edge of r lies on the line $x = x_i$ and its bottom edge lies on the line $y = y_j$. If $j < t$ (or $y_j < y_t$), then $r \notin R_\tau$, so $R_\tau = R_{(p-1,q,t)}$ and $A[p, q, t] = A[p-1, q, t]$. Otherwise, $R_\tau = R_{(p-1,q,t)} \cup \{r\}$. If $r \notin M_\tau$, then again $A[p, q, t] = A[p-1, q, t]$. On the other hand, if $r \in M_\tau$, then none of the other rectangles in R_τ that intersect r can belong to M_τ. There are two cases when $r \in M_\tau$, see Fig. 4. If $i > q$ (or $x_i > x_q$), then let $\tau' = (i-1, q, t)$, otherwise let $\tau' = (i-1, q, j-1)$. It is easy to see that if $r \in M_\tau$, then $M_\tau = M_{\tau'} \cup \{r\}$. Therefore, $A_\tau = A_{\tau'} + 1$. Hence, we obtain the following for $A[p, q, t]$, assuming that $p > q$:

$$A[p,q,t] = \begin{cases} A[p-1,q,t], & \text{no rectangle in } R_{(p,q,t)} \cap R_1 \text{ has the right edge at } x = x_p; \\ \max\big(A[p-1,q,t], A[i-1,q,t]+1\big), & \\ \quad R_{(p,q,t)} \cap R_1 \text{ has a rectangle } r \text{ with the right edge at } x = x_p, \text{ the left edge at} & \\ \quad x = x_i, \text{ and } i > q; & \\ \max\big(A[p-1,q,t], A[i-1,q,j-1]+1\big), & \\ \quad R_{(p,q,t)} \cap R_1 \text{ has a rectangle } r \text{ with the right edge at } x = x_p, \text{ the left edge at} & \\ \quad x = x_i \text{ with } i \leqslant q, \text{ and the bottom edge at } y = y_j. & \end{cases}$$

If there are many rectangles in $R_\tau \cap R_1$ with the right edge on the line $x = x_p$, we divide them into two subsets—the ones whose left edge lies to the left of $x = x_q$ and the ones whose left edge does not lie to the left of $x = x_q$. For each rectangle in the first category, we use the third case and for all rectangles in the second category we apply the second case. We then choose the one that gives the maximum

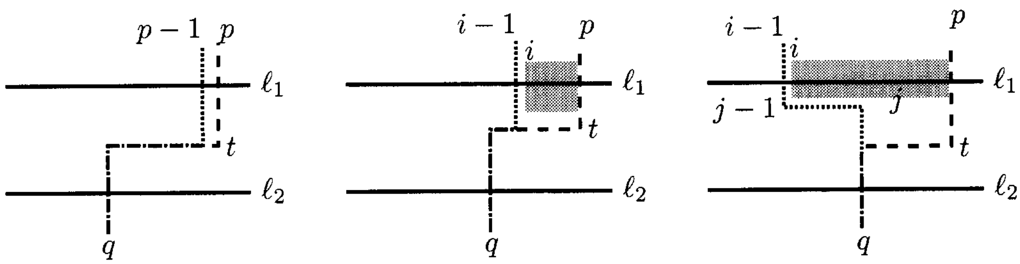

Fig. 4. Filling out the entry $A[p, q, t]$; the three cases.

value. We can fill out the three-dimensional table A in a standard dynamic programming manner [4]. Geometrically, the only constraint on filling out the entries is that when $A[p, q, t]$ is being computed, we must have computed the entries corresponding to the polygonal lines that lie in the closure of the subplane left of $\lambda_{(p,q,t)}$. A straightforward implementation of the dynamic programming requires $O(|R_1 \cup R_2|^3)$ time—most entries take constant time to fill out, except when several rectangles have their right edge at the same p or q. However, we can afford to spend time proportional to the number of rectangles, since the total work still adds up to $O(|R_1 \cup R_2|^3)$.

Let $|R_i| = n_i$, for $i = 1, 2, \ldots, m$, where m is the number of lines used to partition R. Then, clearly $\sum_{i=1}^{m} |R_i| = \sum n_i = n$. In order to compute an independent set of size $2\gamma/3$, we perform the dynamic programming algorithm $m - 1$ times, once for each pair of consecutive lines. Thus the total time complexity is

$$\sum_{i=1}^{m-1} O\big((n_i + n_{i+1})^3\big) = O(n^3).$$

Observe that if $n_i = O(\sqrt{n})$ for all i—a situation that is likely to occur in practice—then the running time is only $O(n^2)$. It is straightforward to adapt the algorithm so that it computes the independent set rather than the size of it.

Theorem 3. *Let R be a set of n unit-height axis-parallel rectangles in the plane. In $O(n^3)$ time, we can compute an independent set of size at least $2\gamma/3$, where γ is the size of a maximum independent set of R.*

Extending the technique to a $(1 + 1/k)$-approximation algorithm is straightforward. We need to compute an optimum solution for the union of rectangles intersecting k consecutive lines. In the dynamic programming algorithm, instead of a 3-dimensional table, we need to fill out a $(2k - 1)$-dimensional table. Geometrically, a $(2k - 1)$-tuple corresponds to a weakly y-monotone, rectilinear polyline with two vertical half-lines, $k - 2$ horizontal edges, and $k - 3$ vertical edges. Each vertical edge has its x-coordinate in X, and each horizontal edge has its y-coordinate in Y. This gives us the polynomial-time approximation scheme with the following performance.

Theorem 4. *Let R be a set of n unit-height axis-parallel rectangles in the plane. In $O(n^{2k-1})$ time, we can compute an independent set of size at least $\gamma/(1 + 1/k)$, for any $k \geqslant 1$, where γ is the size of a maximum independent set of R.*

5. Conclusions

We have given approximation algorithms for computing maximum size non-intersecting subset in sets of rectangles. The work is motivated from label placement at points, where the rectangles represent the bounding boxes of labels. The approximation scheme was known for the restrictive case of unit size square labels [11], which occurs for fixed precision decimal numbers as labels. We gave a different approximation scheme for unit height labels with varying widths, which is the standard situation for labels that are names, or labels of different type with fixed font size.

The algorithms for labeling support the situation where several positions for the label of any point are allowed. The restriction is that all positions of the label of a point intersect each other. Also, the asymptotic running time is not affected if a constant number of positions is allowed for each label.

The maximum non-intersecting subset of rectangles problem can be seen as a maximum independent set problem in a special type of graph. The approximation algorithm we presented for these graphs is considerably better than what is theoretically possible for general graphs. However, we were not able to obtain a polynomial time, constant-factor approximation algorithm for the case of arbitrary axis-parallel rectangles. This is an interesting open problem.

Acknowledgements

The authors thank Alexander Wolff for helpful comments.

References

[1] M. Bellare, M. Sudan, Improved non-approximability results, in: Proc. 26th Annu. ACM Sympos. Theory Comput., 1994, pp. 184–193.

[2] J. Christensen, S. Friedman, J. Marks, S. Shieber, Empirical testing of algorithms for variable-sized label placement, in: Proc. 13th Annu. ACM Sympos. Comput. Geom., 1997, pp. 415–417.

[3] J. Christensen, J. Marks, S. Shieber, An empirical study of algorithms for point-feature label placement, ACM Trans. Graph. 14 (1995) 202–232.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.

[5] S. Doddi, M. Marathe, A. Mirzaian, B. Moret, B. Zhu, Map labeling and its generalization, in: Proc. 8th ACM-SIAM Sympos. Discrete Algorithms, 1997, pp. 148–157.

[6] J.S. Doerschler, H. Freeman, A rule-based system for dense-map name placement, Commun. ACM 35 (1992) 68–79.

[7] M. Formann, F. Wagner, A packing problem with applications to lettering of maps, in: Proc. 7th Annu. ACM Sympos. Comput. Geom., 1991, pp. 281–288.

[8] R.J. Fowler, M.S. Paterson, S.L. Tanimoto, Optimal packing and covering in the plane are NP-complete, Inform. Process. Lett. 12 (3) (1981) 133–137.

[9] H. Freeman, Computer name placement, in: D.J. Maguire, M.F. Goodchild, D.W. Rhind (Eds.), Geographical Information Systems: Principles and Applications, Longman, London, 1991, pp. 445–456.

[10] D.S. Hochbaum, W. Maas, Approximation schemes for covering and packing problems in image processing and VLSI, J. ACM 32 (1985) 130–136.

[11] H.B. Hunt III, M.V. Marathe, V. Radhakrishnan, S.S. Ravi, D.J. Rosenkrantz, R.E. Stearns, A unified approach to approximation schemes for NP- and PSPACE-hard problems for geometric graphs, in: Proc. 2nd Europ. Symp. on Algorithms, Lecture Notes in Computer Science, Vol. 855, 1995, pp. 424–435.

[12] H. Imai, T. Asano, Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane, J. Algorithms 4 (1983) 310–323.
[13] L. Kučera, K. Mehlhorn, B. Preis, E. Schwarzenecker, Exact algorithms for a geometric packing problem, in: Proc. 10th Sympos. Theoret. Aspects Comput. Sci., Lecture Notes in Computer Science, Vol. 665, Springer, New York, 1993, pp. 317–322.
[14] M.V. Marathe, H. Breu, H.B. Hunt III, S.S. Ravi, D.J. Rosenkrantz, Simple heuristics for unit disk graphs, Networks 25 (1995) 59–68.
[15] F. Wagner, A. Wolff, An efficient and effective approximation algorithm for the map labeling problem, in: Proc. 3rd Annu. European Sympos. Algorithms, Lecture Notes in Computer Science, Vol. 979, Springer, New York, 1995, pp. 420–433.
[16] F. Wagner, A. Wolff, Map labeling heuristics: Provably good and practically useful, in: Proc. 11th Annu. ACM Sympos. Comput. Geom., 1995, pp. 109–118.