Computational Geometry ••• (••••) •••–•••

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Matching points with rectangles and squares ☆

Sergey Bereg [a], Nikolaus Mutsanas [b], Alexander Wolff [c,*]

[a] *Department of Computer Science, University of Texas at Dallas, USA*
[b] *Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Switzerland*
[c] *Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, The Netherlands*

ARTICLE INFO

Article history:
Received 11 February 2007
Received in revised form 12 February 2008
Accepted 20 May 2008

Communicated by F. Hurtado

Keywords:
Matching with geometric objects
Weak and strong matching
Perfect matching
NP-hardness
Approximation

ABSTRACT

In this paper we deal with the following natural family of geometric matching problems. Given a class $\mathcal{C}$ of geometric objects and a set P of points in the plane, a $\mathcal{C}$-matching is a set $M \subseteq \mathcal{C}$ such that every $C \in M$ contains exactly two elements of P. The matching is *perfect* if it covers every point, and *strong* if the objects do not intersect. We concentrate on matching points using axes-aligned squares and rectangles.

We propose an algorithm for strong rectangle matching that, given a set of n points, matches at least $2\lfloor n/3 \rfloor$ of them, which is worst-case optimal. If we are given a combinatorial matching of the points, we can test efficiently whether it has a realization as a (strong) square matching. The algorithm behind this test can be modified to solve an interesting new point-labeling problem. On the other hand we show that it is NP-hard to decide whether a point set has a perfect strong square matching.

1. Introduction

The problem of matching points with geometric objects is an attempt to generalize the notion of a graph-theoretical matching to geometric environments. Regarding edges of a geometric graph as line segments is a first step towards matching with geometric objects. Instead of using line segments to match points, a matching can be defined by elements of some family of geometric objects (like squares or disks) that contain exactly two of the input points. This class of geometric matching problems has recently been introduced by Ábrego et al. [1].

A geometric matching is called *strong*, if the geometric objects do not intersect. We refer to any geometric matching as *weak*, that is, if we do not know whether it is strong. Similar to matchings in graphs, we call a geometric matching *perfect*, if it covers every point of the point set. We describe the general problem and give a summary of previous results in Section 2. In this paper we deal with the problem of matching points with axes-aligned (closed) rectangles and squares.

As it turns out, matching with these geometric objects has strong links to map labeling, the problem of annotating graphical features on maps and other diagrams. On the one hand, one of our matching algorithms uses a map-labeling algorithm as a subroutine; on the other hand, the same matching algorithm solves a new and interesting map-labeling problem.

☆ This work has been supported by grant WO 758/4-2 of the German Research Foundation (DFG) and by the Swiss National Science Foundation project 200021-104017/1, "Power Aware Computing". It is based on the preliminary version [S. Bereg, N. Mutsanas, A. Wolff, Matching points with rectangles and squares, in: J. Wiedermann, J. Stuller, G. Tel, J. Pokorny, M. Bieliková (Eds.), Proceedings 32nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'06), in: Lecture Notes in Computer Science, vol. 3831, Springer-Verlag, Berlin, 2006, pp. 177–186].

* Corresponding author.
E-mail addresses: besp@utdallas.edu (S. Bereg), nikolaus@idsia.ch (N. Mutsanas).
URL: http://www.win.tue.nl/~awolff (A. Wolff).

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2008.05.001

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

For an overview over the area of map labeling, see the map-labeling bibliography [25]. The two most important conditions for placing labels are (a) that labels do not overlap and (b) that labels are in the vicinity of the objects they annotate (or visually connected to them by other means such as arrows [4]). As a consequence, it may not be possible to label all objects on a map with labels of the given sizes. Hence one is either interested in labeling as many objects as possible or in maximizing a scaling factor σ such that scaling all labels by σ allows to label all objects. The problem categories connected to these two objective functions are called *label-number maximization* and *label-size maximization*, respectively. A generalization of the former objective function is considered if objects have weights; then one is usually interested in maximizing the sum of the weights of the labeled objects [9]. Another way to categorize map-labeling problems is according to the dimension of the objects that are to be labeled: point labeling versus (polygonal) line labeling versus area labeling. In this paper we will use an algorithm for labeling line segments as a subroutine for an algorithm that matches points with squares and in turn we will use that algorithm for labeling points with squares or rectangles.

One further distinguishes between *fixed-position models* and *slider models*. In fixed-position models, each label has a predetermined finite set of *anchor points* on its boundary (for example, the four corner points), and the label must be placed so that one of its anchor points coincides with the point to be labeled. In slider models, the anchor points form *anchor segments* on the boundary of the label (for example, its bottom edge). Van Kreveld et al. [24] introduced a taxonomy of fixed-position and slider models, which was later refined by Poon et al. [19].

Matching with rectangles. Clearly, any point set in general position (that is, no two points have the same x- or y-coordinate) has a strong perfect rectangle matching. If we drop the general-position assumption, however, there are point sets where considerably less points can be rectangle-matched. In fact, there is a family of point sets $(P_k)_{k\geqslant 1}$ such that P_k contains $n = 3k + 2$ points of which at most $2k = 2\lfloor n/3 \rfloor$ can be strongly rectangle-matched, see Section 3. We complement this upper bound construction by a matching lower bound; an algorithm that matches at least $2\lfloor n/3 \rfloor$ points in any given set of n points. Obviously our algorithm is a $\approx 2/3$-approximation for the problem of finding a strong rectangle matchings of maximum cardinality. The question remains open whether this problem can be solved exactly in polynomial time.

Matching with squares. Next, we consider the problem of matching points with axes-aligned squares. We give efficient algorithms (see Section 4) that decide the following: given a point set and a combinatorial matching of the points, can the matching be realized by a weak or by a strong square matching? The algorithm for weak square matching runs in optimal $O(n \log n)$ time, the algorithm for strong square matchings takes $O(n^2 \log n)$ time. The latter algorithm uses an algorithm of Strijk and van Kreveld [23] for labeling axis-parallel line segments as a subroutine.

Application to map labeling. We show that our algorithm for strong square matching in turn helps solving an interesting new map-labeling problem, see Section 5. In our new problem, we are given a set of points in the plane and for each point a rectangular axes-aligned label. One of the four edges of the label (top, bottom, left, or right) is marked. Label sizes and marks can vary from point to point. The aim is to label all points such that no two labels intersect and each label touches its point with the marked edge. Our results for this problem are as follows. We give an algorithm that solves the decision problem in $O(n^2 \log n)$ time. If all points can be labeled simultaneously, the same algorithms also yields a placement of the labels. If labels are congruent axes-aligned squares, we can solve the corresponding label-size maximization problem. In other words, we can compute the maximum scaling factor $\sigma_{\max} > 0$ such that scaling all labels by $\sigma_{\max}$ yields a feasible instance (scaling the labels by any $\sigma > \sigma_{\max}$ yields an infeasible instance). Our algorithm for label-size maximization takes $O(n^2 \log^2 n)$ time.

Label-size maximization has been considered for fixed-position models. For example, Wagner and Formann [10] have given an $O(n \log n)$-time 2-approximation for the case that labels are congruent squares, and Jung and Chwa have given an $O(n^2 \log n)$-time 2-approximation for the case that labels are arbitrary rectangles [12]. They solve the corresponding decision problem in $O(n \log n)$ time. Label-size maximization has also been considered for the case of labels that are congruent disks [11] and for the case that each points receives two disk-shaped labels [11], or two or three (possibly sliding) square labels [7, 21]. Except for the problem of placing three square labels per point, all the problems we have mentioned in this paragraph are NP-hard and cannot be approximated arbitrarily well (assuming $\mathcal{P} \neq \mathcal{NP}$).

Graphically speaking, our new labeling model lies on the "boundary" of what can be computed efficiently (assuming $\mathcal{P} \neq \mathcal{NP}$). Van Kreveld et al. [24] have shown that labeling points with sliding labels is NP-hard if any point on the label boundary is an anchor point, even if all labels are squares of the same size. In other words, our new point-labeling problem becomes hard if we drop the requirement that each point "knows" which label edge it must touch. An inspection of their hardness proof shows that the problem remains hard even if we specify for each point whether its label is allowed to slide horizontally or vertically. This corresponds to marking *two* (opposite) edges of a label in the input and to requiring as above that each point is touched by a marked edge of its label.

Complexity. Finally, we show that it is NP-hard to decide whether a given point set admits a perfect strong square matching, see Section 6.

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

2. Problem definition and previous work

Following Ábrego et al. [1] we define the problem formally as follows:

Definition 1. Let $\mathcal{C}$ be a family of geometric objects and let P be a set of n points with n even. A $\mathcal{C}$-matching of P is a set $M \subseteq \mathcal{C}$, such that every $C \in M$ contains exactly two points of P. A $\mathcal{C}$-matching M is called *strong* if no two elements of M intersect. Arbitrary $\mathcal{C}$-matchings are called *weak*. A matching M of P is called *perfect* if every $p \in P$ is contained in some $C \in M$.

For example, if $\mathcal{S}$ is the family of line segments, a strong perfect $\mathcal{S}$-matching can be computed in $O(n \log n)$ time by sorting the input points lexicographically and putting every second pair of consecutive points in the matching. Rendl and Woeginger [22] have investigated the family $\mathcal{A}$ of axis-aligned line segments. They gave an algorithm that decides in $O(n \log n)$ time whether a set of n points has a perfect weak $\mathcal{A}$-matching and, if so, computes such a matching. They also showed that it is NP-hard to decide whether a set of points has a perfect strong $\mathcal{A}$-matching.

Dumitrescu and Steiger [8] have also considered strong matchings with line segments. Instead of restricting the slopes of the segments, they assume that each point is either black or white, and only points of the same color can be matched. They give an algorithm that matches at least $5n/6 - O(1)$ points in any n-point set, where n and the sizes of the two color classes are even. The algorithm runs in $O(n \log n)$ time. They also show that there are n-point sets where at least $n/156$ points cannot be matched.

The link between a matching with geometric objects and a graph-theoretical matching is established by the following definition:

Definition 2. Let $\mathcal{C}$ be a family of geometric objects and P a point set in the plane. The matching graph for $\mathcal{C}$ and P is the graph $G_{\mathcal{C}}(P, E_{\mathcal{C}})$, where two nodes $p \neq q$ are adjacent if and only if there is an object $C \in \mathcal{C}$ that contains exactly these two points, that is, $C \cap P = \{p, q\}$. We regard a geometric matching as a *realization* of the underlying combinatorial matching.

Note that a maximum-cardinality matching in $G_{\mathcal{C}}$ yields weak maximum-cardinality matchings for any family $\mathcal{C}$ of geometric objects. Using the algorithm of Micali and Vazirani [17] this takes $O(m\sqrt{n})$ time, where m is the number of edges of $G_{\mathcal{C}}$. Thus a weak maximum-cardinality matching can be computed in $O(n^{2.5})$ time if $G_{\mathcal{C}}$ is given.

The problem of matching points with geometric objects was introduced by Ábrego et al. [1]. Their results are based on the assumption that the points are in general position, that is, there are no two points with the same x- or y-coordinate. Ábrego et al. showed that such point sets always have a weak perfect matching with axes-aligned squares, provided that n is even. They also showed that any set P of n points has a strong square matching that covers at least $2\lceil n/5 \rceil$ points. If the point set is additionally in convex position, a perfect strong square matching always exists, provided that n is even. On the other hand, Ábrego et al. gave arbitrarily large point sets such that any strong matching covers at most a fraction of $12/13$ of the points. Recently they managed to lower this fraction to $10/11$ [2].

For the family $\mathcal{D}$ of disks, Ábrego et al. also showed that a weak perfect matching always exists, provided that n is even. They prove this assumption by using the matching graph $G_{\mathcal{D}}$ with respect to $\mathcal{D}$. By definition, two points $p, q \in P$ are adjacent in $G_{\mathcal{D}}$ if and only if there is a disk that contains exactly those two points. This is equivalent to p and q being neighbors in the Delaunay triangulation of P. Dillencourt [6] proved that for n even the Delaunay triangulation always contains a perfect matching. Ábrego et al. also showed that any set of n points has a strong disk matching that covers at least $2\lceil (n-1)/8 \rceil$ points. On the other hand, they constructed arbitrarily large point sets such that any strong disk matching covers at most a fraction of $72/73$ of the points.

3. Rectangle matching

If points are in general position, the problem of matching points with axes-aligned rectangles is trivial. An obvious algorithm that yields a perfect strong rectangle matching is the following: Sort the points lexicographically and cover each point with odd index and its successor by their bounding box. Since the order of the x-coordinates is strictly monotone, boxes do not overlap.

Equally simple is the problem in which the orientation of the rectangles can be chosen arbitrarily. If the point set is not in general position, there is an angle ε by which we can tilt the point set so that the point set is in general position.

However, if we drop the condition of general position, the problem of matching points with axes-aligned rectangles becomes interesting. For $k = 4$ Fig. 1 depicts the point set $P_k = \{(i, i), (i-1, i), (i, i-1) \mid 1 \leqslant i \leqslant k\} \cup \{(k, k+1), (k+1, k)\}$ and its matching graph G_k, which has $n = 3k + 2$ vertices and $4k$ edges. Each of the *central* nodes $(1, 1), \ldots, (k, k)$ has degree 4 in G_k, and each edge is incident to a central node. Clearly, only one of the edges incident to a central node can be in a matching. Thus a maximum rectangle matching of P_k has cardinality k; out of $n = 3k + 2$ points it covers $2\lfloor n/3 \rfloor = 2k$. This shows that $2/3$ is an upper bound for the ratio of points that can always be covered by a rectangle matching.

We now present an efficient algorithm that always yields a strong rectangle matching that covers at least $2\lfloor n/3 \rfloor$ points in an n-element point set, for $n \geqslant 3$. We are going to use the following notation. For two points a and b we write $a < b$ if

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

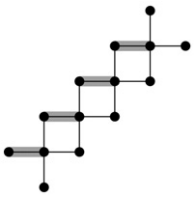


Fig. 1. Set of n points of which at most $2\lfloor n/3 \rfloor$ can be matched.

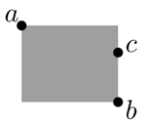


Fig. 2. The point c blocks (a, b).

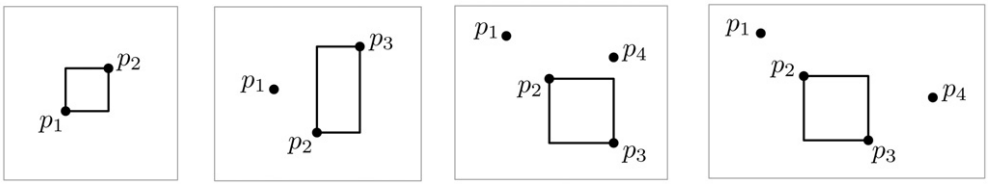


Fig. 3. The rectangle R in the proof of Lemma 1.

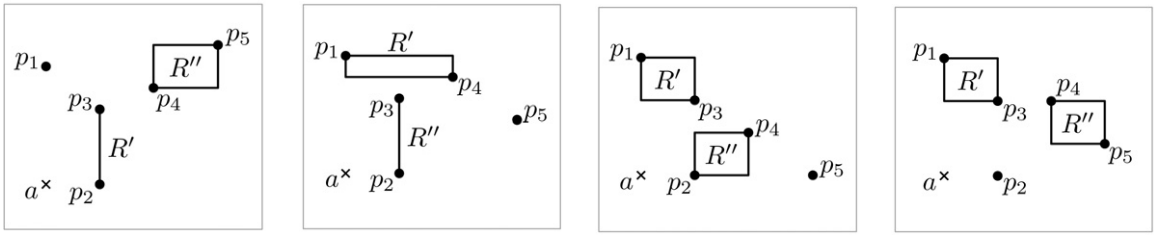


Fig. 4. Rectangles R' and R'' in the proof of Lemma 2.

and only if a precedes b in the lexicographic order. We define $a > b$, $a \leqslant b$, and $a \geqslant b$ analogously. For two sets $A, B \subseteq \mathbb{R}^2$, we write $A < B$ if $a < b$ for each $a \in A$ and $b \in B$. Again we define $A > B$, $A \leqslant B$, $A \geqslant B$ analogously. We also use $a < B$ if $a < b$ for any $b \in B$.

Let $R(a, b)$ denote the bounding box of a and b, that is, the smallest axes-aligned rectangle containing the points a and b. Let $a < b < c$ be three points in the plane. We say that c *blocks* the pair (a, b) if $c \in R(a, b)$. Due to the order of the points this can only happen if b lies to the right and below a, and c lies on the right edge of $R(a, b)$. For an example, see Fig. 2.

Lemma 1. *Let $S = (p_1, \ldots, p_4)$ be a sequence of four distinct points in lexicographic order. Either*

(i) *there is a rectangle R with the property that* (a) *exactly two of the points $\{p_1, p_2, p_3\}$ lie in R and* (b) *it holds that $p_1 \leqslant R < p_4$, or*
(ii) *$R(p_2, p_3) \cap S = \{p_2, p_3, p_4\}$ and p_4 blocks (p_2, p_3).*

Proof. Let $p_i = (x_i, y_i)$ for $i = 1, \ldots, 4$. If $y_2 \geqslant y_1$ then $R = R(p_1, p_2)$ satisfies (i), see Fig. 3(a). So we assume that $y_2 < y_1$. Note that this implies $x_2 > x_1$ and thus $p_1 < R(p_2, p_3)$. Set $R = R(p_2, p_3)$.

First suppose that $y_3 \geqslant y_2$. Then R satisfies (i) since $p_1 < R \leqslant p_3 < p_4$, see Fig. 3(b).

Now suppose that $y_3 < y_2$. If $y_4 > y_2$ or $x_4 > x_3$ then R again satisfies (i), see Fig. 3 (c) and (d). Otherwise (that is, $y_4 \leqslant y_2$ and $x_4 = x_3$), $R \cap S = \{p_1, p_2, p_3\}$ and p_4 blocks (p_2, p_3). The lemma follows. □

Lemma 2. *Let $S = (p_1, \ldots, p_6)$ be a sequence of six distinct points in lexicographic order such that p_3 blocks (p_1, p_2). Let a be the lower-left corner of $R(p_1, p_2)$. There are two disjoint rectangles R' and R'' such that $|R' \cap S| = 2$, $a \leqslant R' \cup R''$ and either*

(i) *$|R'' \cap S| = 2$ and $R' \cup R'' < p_6$, or*
(ii) *$R'' = R(p_4, p_5)$, $R'' \cap S = \{p_4, p_5, p_6\}$, and p_6 blocks (p_4, p_5).*

Proof. If $y_5 \geqslant y_4$ then $R' = R(p_2, p_3)$ and $R'' = R(p_4, p_5)$ satisfy (i), see Fig. 4(a). So we can assume that $y_5 < y_4$. In this case the ordering of S implies $x_5 > x_4$. Now we distinguish three cases depending of the relation of y_3 and y_4.

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

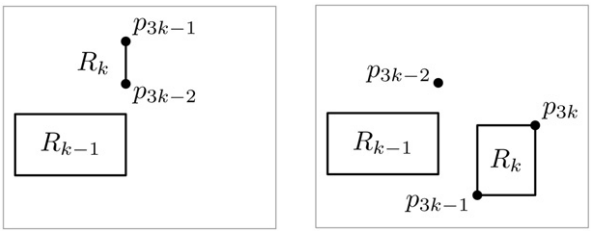


Fig. 5. Choice of rectangle R_k in the non-blocking case.

(a) $y_3 < y_4$.
Then $R' = R(p_1, p_4)$ and $R'' = R(p_2, p_3)$ satisfy (i), see Fig. 4(b).
(b) $y_3 > y_4$.
Then $R' = R(p_1, p_3)$ and $R'' = R(p_2, p_4)$ satisfy (i), see Fig. 4(c).
(c) $y_3 = y_4$.
Then $R(p_1, p_3)$ and $R(p_4, p_5)$ are disjoint, see Fig. 4(d). If $R(p_4, p_5) < p_6$ then $R' = R(p_1, p_3)$ and $R'' = R(p_4, p_5)$ satisfy (i). Otherwise (that is, if $R(p_4, p_5) \not< p_6$), p_6 blocks (p_4, p_5), and $R' = R(p_1, p_3)$ and $R'' = R(p_4, p_5)$ satisfy (ii).

Now the lemma follows. □

Theorem 1. *Any set of n points in the plane has a strong rectangle matching that matches at least $2\lfloor n/3 \rfloor$ points. Such a matching can be computed in $O(n \log n)$ time.*

Proof. Let $P = \{p_1, \ldots, p_n\}$ be a set of n points sorted in lexicographic order. Let $k = \lfloor n/3 \rfloor$ be the number of matching rectangles we aim for. If n is not a multiple of 3, then there are $3k+1$ or $3k+2$ points, but since we never use the points p_{3k+1} or p_{3k+2}, we may as well assume that $n = 3k$. We process triplets $(p_{3i-2}, p_{3i-1}, p_{3i})$ of consecutive points in P for $i = 1, \ldots, k$ and show that there is a strong rectangle matching $R_1, \ldots, R_k$. To facilitate counting we charge the ith rectangle R_i to the ith triplet $(p_{3i-2}, p_{3i-1}, p_{3i})$. There are two cases for processing a triplet. (For $i = 1$ the first case applies.)

Non-blocking condition for i. Suppose that $i-1$ matching rectangles $R_1, \ldots, R_{i-1}$ have already been computed and that it holds that $R_j < p_{3i+1}$ for $j = 1, \ldots, i-1$. We apply Lemma 1 to the sequence $(p_{3i-2}, p_{3i-1}, p_{3i}, p_{3i+1})$. If case (i) of Lemma 1 applies, then there is a rectangle R that contains exactly two of the points $p_{3i-2}, p_{3i-1}, p_{3i}$ (as in Fig. 3, where the situation is shown for $i = 1$). Let $R_i = R$. Since $R_i < p_{3i+1}$, the non-blocking condition holds for $i+1$. On the other hand, if case (ii) of Lemma 1 applies, the triplet $(p_{3i-2}, p_{3i-1}, p_{3i})$ remains temporarily uncharged and p_{3i+1} blocks (p_{3i-1}, p_{3i}). This situation is described next.

Blocking condition for i. Suppose that $i-2$ matching rectangles $R_1, \ldots, R_{i-2}$ have been computed such that each is disjoint from $R(p_{3i-4}, p_{3i-3})$. Further, suppose that p_{3i-2} blocks (p_{3i-4}, p_{3i-3}). Now we apply Lemma 2 to the sequence $S = (p_{3i-4}, p_{3i-3}, p_{3i-2}, p_{3i-1}, p_{3i}, p_{3i+1})$. This yields two disjoint rectangles R' and R''. The rectangle R' contains exactly two points of the sequence S. Let $R_{i-1} = R'$. If case (i) of Lemma 2 applies, let $R_i = R''$. Case (i) guarantees that $R' \cup R'' < p_{3i+1}$, which means that we have the non-blocking condition for $i+1$. On the other hand, if case (ii) of Lemma 2 applies, the triplet $(p_{3i-2}, p_{3i-1}, p_{3i})$ remains temporarily uncharged and p_{3i+1} blocks (p_{3i-1}, p_{3i}). Thus, we have the blocking condition for $i+1$.

Let m be the number of matching rectangles found when this process stops. If $m = k$, the first claim of the theorem follows. If $m < k$, we show that the number of matching rectangles can be increased to k. There are two cases similar to those above, but here Lemmas 1 and 2 cannot be applied since there is no point p_{n+1}.

Non-blocking case. Here $m = k-1$ matching rectangles $R_1, \ldots, R_{k-1}$ have already been computed, and it holds that $R_j < p_{3k-2}$ for $j = 1, \ldots, k-1$. If $x_{3k-2} = x_{3k-1}$, we let $R_k = R(p_{3k-2}, p_{3k-1})$, see Fig. 5(a). Otherwise $x_{3k-2} < x_{3k-1}$, and we let $R_k = R(p_{3k-1}, p_{3k})$, see Fig. 5(b). Clearly in both cases $p_{3k-2} \leqslant R_k$, and thus R_k is disjoint from $R_1, \ldots, R_{k-1}$.

Blocking case. Here $m = k-2$ matching rectangles $R_1, \ldots, R_{k-2}$ have been computed, each is disjoint from $R(p_{3k-4}, p_{3k-3})$, and p_{3k-2} blocks (p_{3k-4}, p_{3k-3}). Let $R_{k-1} = R(p_{3k-3}, p_{3k-2})$. See Fig. 6(a). Note that R_{k-1} is contained in $R(p_{3k-4}, p_{3k-3})$ and thus disjoint from each of $R_1, \ldots, R_{k-2}$.

Suppose $x_{3k-2} = x_{3k-1}$. This implies $y_{3k-2} < y_{3k-1}$. Let $R_k = R(p_{3k-4}, p_{3k-1})$. If $y_{3k-4} \geqslant y_{3k-1}$ then R_k is contained in $R(p_{3k-4}, p_{3k-3})$, see Fig. 6(b). Otherwise R_k touches and lies vertically above $R(p_{3k-4}, p_{3k-3})$, see Fig. 6(c). For the latter case observe that no point can lie vertically above p_{3k-4} (since p_{3k-3} does not). Thus in either case any rectangle that intersects R_k would also intersect $R(p_{3k-4}, p_{3k-3})$. Therefore, R_k is disjoint from each of $R_1, \ldots, R_{k-2}$, but also from R_{k-1} since $R_{k-1} \leqslant p_{3k-2}$ and p_{3k-2} lies strictly below R_k.

Now suppose $x_{3k-2} < x_{3k-1}$. Then let $R_k = R(p_{3k-1}, p_{3k})$. See Fig. 6(d). In this case $R_k > R_{k-1}$ and thus R_k is disjoint from each of $R_1, \ldots, R_{k-1}$.

Our proof is constructive. After sorting the points in P lexicographically, the cases of Lemmas 1 and 2 can be processed in constant time each, thus proving the second claim of the theorem. The theorem follows. □

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

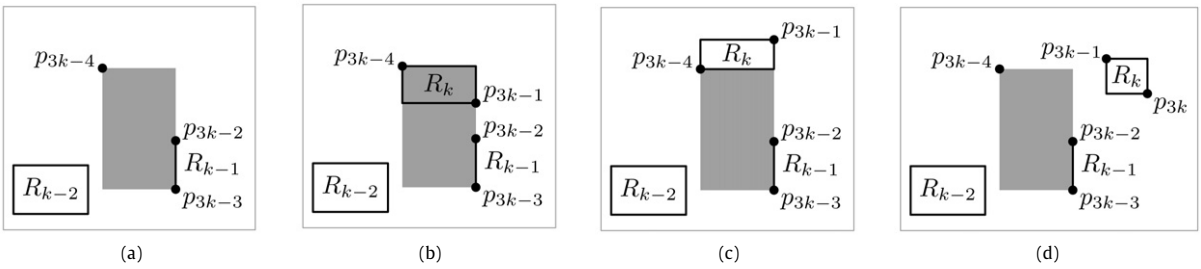


Fig. 6. Choice of rectangles R_{k-1} and R_k in the blocking case.

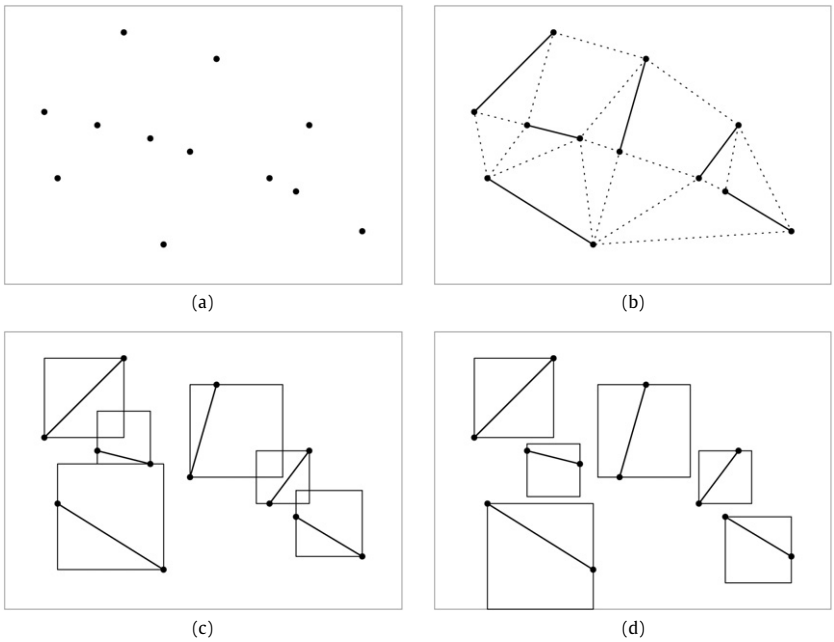


Fig. 7. Realizations of a combinatorial matching by a weak and a strong square matching. (a) Point set P, (b) matching graph of P; solid edges: combinatorial matching M of P, (c) weak square realization of M, (d) strong square realization of M.

The point set in Fig. 1 shows that the bound in Theorem 1 is tight. Applying the algorithm from the proof of Theorem 1 to the point set in Fig. 1 yields the matching that consists of the edges drawn bold. It remains open, however, how to compute strong rectangle matchings of maximum cardinality for individual point sets.

As mentioned in the previous section, we can compute weak matchings of maximum cardinality in $O(n^{2.5})$ time given the corresponding matching graph. For rectangles, we can slightly improve on this. Consider the special case where no two input points have the same x-coordinate or no two input points have the same y-coordinate. In this case we can find a perfect weak rectangle matching given a set of n points in $O(n \log n)$ time. This idea can be generalized as follows.

Proposition 1. *Given a set P of n points, a weak rectangle matching of maximum cardinality can be computed in $O(\beta n^{1.5})$ time, where β is the minimum of the number of different x-coordinates and the number of different y-coordinates in P.*

Proof. Observe that each vertex in the matching graph G_{rect} of P has degree at most 2β, and thus applying the maximum-cardinality matching algorithm of Micali and Vazirani [17] to G_{rect} yields the desired time bound. □

4. Square matching

Note that, contrary to rectangle matchings, a square matching is not uniquely defined by a given combinatorial matching. Fig. 7 shows a weak and a strong square realization of the same combinatorial matching. In this section we present efficient algorithms that decide whether a given combinatorial matching $M \subseteq \binom{P}{2}$ of a point set P has a weak or a strong square realization, where $\binom{P}{2} = \{\{p, q\} \mid p, q \in P, p \neq q\}$ is the set of all unordered pairs of points in P.

Consider a square matching for a given point set P. Let the squares of this matching shrink as much as possible while still covering the points. The resulting squares are of minimum size among all squares that contain the two points, and

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

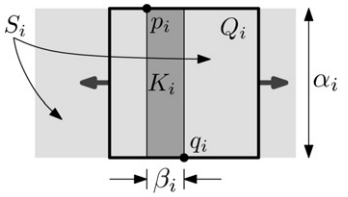


Fig. 8. A point pair $\{p_i, q_i\}$ with a minimal (horizontally sliding) square Q_i (bold), its sliding space S_i (light gray) and its kernel K_i.

the points now lie on the square boundary. This new matching consists of squares that are contained in the squares of the initial matching. This means that when deciding whether a given matching can be realized as a square matching, it suffices to examine square matchings where all the squares are of minimum size.

Let Q_i be a minimum-size square that contains two given points p_i and q_i. It is easy to see that the edge length α_i of a minimal square is the distance of the two points in the maximum (or L_∞-) metric, see Fig. 8. If the two coordinate differences are not equal, the square can be slid—to some extend—in the direction of the smaller coordinate difference β_i. This leads to the model of the sliding squares illustrated in Fig. 8.

The *kernel* K_i of a point pair $\{p_i, q_i\}$ is their bounding box, that is, the smallest axes-aligned rectangle that contains the two points. The kernel consists of the part of the plane that is contained in *every* (minimal) square that contains the two points. In other words, the kernel is the intersection of these squares. We define the *sliding space* S_i of $\{p_i, q_i\}$ to be the union of those squares minus the kernel, see Fig. 8. We say that the (minimal) square Q_i slides *horizontally* if $\beta_i < \alpha_i$ (see Fig. 8), otherwise Q_i slides *vertically*.

Note that the kernel degenerates to an axis-parallel line segment if the two points share a coordinate, and that the sliding space vanishes if the two points lie on a line of slope $+1$ or -1. Then the minimal square that contains the two points is uniquely defined. In spite of this we will consider such squares to be vertically sliding. In what follows, the *position* of a vertically sliding square Q_i always corresponds to the y-coordinate of its bottom edge and the position of a horizontally sliding square correspond to the x-coordinate of its left edge. Let Q be a minimal square that contains p and q. If at least one of the two points lies in a corner of Q, we say that Q is in *extremal position*.

4.1. Weak square realizations

Now it is easy to give an algorithm that checks whether a given matching M of a point set P has a *weak* square realization: for each point pair $\{p, q\}$ in M we compute kernel and sliding space. If the kernel contains input points other than p or q, then M does not have a square realization. Otherwise we check whether there are input points in both connected components of the sliding space. If not, we can place a square matching p and q into the union of the kernel and the empty component. If both components contain input points, we compute in each component the point closest to the kernel. We call the resulting points a and b. If the L_∞-distance of a and b is larger than the L_∞-distance of p and q, then we can place a square that contains the kernel and matches p and q anywhere between a and b. Otherwise, if the L_∞-distance of a and b is at most that of p and q, M does not have a square realization. Using priority search trees [16], this algorithm can be implemented to run in $O(n \log n)$ time.

Theorem 2. *Given a set P of n points and a combinatorial matching $M \subseteq \binom{P}{2}$, it can be decided in $O(n \log n)$ time whether M has a weak square realization.*

A matching lower bound on the running time can be achieved by reduction from the Element Uniqueness problem (also known as Element Distinctness problem), which consists of deciding whether or not a sequence $(a_1, a_2, \ldots, a_n)$ of real numbers contains two equal elements. Ben-Or [5] has shown that the time complexity of this problem has a lower bound of $\Omega(n \log n)$ in the algebraic computation-tree model. This result was later strengthened by Yao [26], who showed that the same lower bound holds even if all numbers in the sequence are integers.

Theorem 3. *Given a set P of n points and a combinatorial matching $M \subseteq \binom{P}{2}$, deciding whether M has a weak square realization takes $\Omega(n \log n)$ time in the algebraic computation-tree model.*

Proof. We reduce the integral version of the Element Uniqueness problem to the weak square matching problem. Given a sequence $A = (a_1, a_2, \ldots, a_n)$ of integers, we construct a set $P = \{p_1, \ldots, p_n, q_1, \ldots, q_n\}$ of $2n$ distinct points on the x-axis as follows. For each $a_i \in A$ let $p_i = (a_i + 1/2^{n+i}, 0)$ and $q_i = (a_i + 1/2^i, 0)$. Finally, let $M = \{\{p_1, q_1\}, \ldots, \{p_n, q_n\}\}$ be a (perfect) combinatorial matching of P. We now show that M has a weak square realization if and only if all elements of A are distinct.

For the "if" part we assume that the elements in the sequence A are pairwise distinct. Then it is obvious that M has a weak square realization: for each $i = 1, \ldots, n$, let S_i be a square of side length $1/2$ whose left edge contains the point

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

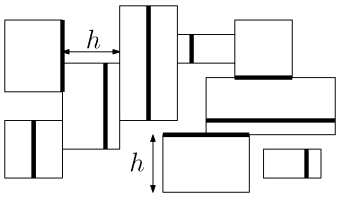


Fig. 9. A set of line segments (bold) with a height-h rectangle labeling (thin).

$(a_i, 0)$. Then it is clear that S_i contains the line segment $\overline{p_i q_i}$. Due to the fact that the elements of A are integers and that the squares have side length 1/2, it is clear that each square contains exactly two points of P. (Actually, no two squares intersect, that is, the realization is even strong.)

For the "only if" part we assume that M has a weak square realization. Let $S = \{S_1, S_2, \ldots, S_n\}$ be a weak square matching of P that realizes M, that is, for $i = 1, \ldots, n$ the square S_i matches p_i and q_i. Now suppose that the sequence A contains two elements a_i and a_j with $a_i = a_j$ and $i < j$. Since S_i matches $\{p_i, q_i\}$, the line segment $\overline{p_i q_i}$ is contained in S_i. Due to $i < j < n + i$, we conclude that $a_i + 1/2^{n+i} < a_i + 1/2^j = a_j + 1/2^j < a_i + 1/2^i$ and thus $q_j \in \overline{p_i q_i} \subset S_i$. This contradicts the fact that S is a weak square matching.

Obviously, the reduction takes linear time. The theorem follows. □

4.2. Strong square realizations

Now we turn to the problem of finding a *strong* square realization for a given combinatorial matching. We first note (in the following corollary to Theorem 3) that the lower bound for the time complexity of computing a strong square realization is the same as that of computing a weak square realization.

Corollary 1. *Given a set P of n points and a combinatorial matching $M \subseteq \binom{P}{2}$, deciding whether M has a strong square realization takes $\Omega(n \log n)$ time in the algebraic computation-tree model.*

Proof. In the proof of Theorem 3 we showed that the matching M has a weak square realization if and only if all elements in the set A are distinct. In the argument for the "if" part we already observed that the elements in A being distinct yields a square realization that is even strong. In the "only if" part we saw that two equal elements in A exclude any weak and thus also any strong square realization. This completes our proof. □

Due to the observations at the beginning of this section, it suffices to examining combinations of placements of squares of *fixed size*. The idea behind our algorithm for solving the strong realization problem is that instead of considering a combination among all possible positions of the squares, we need only check combinations among a few relevant positions for each square. The correctness of the algorithm follows if we prove that the existence of a strong realization implies the existence of a strong realization among the combinations that we consider.

It turns out that a map-labeling problem is related to our problem, namely the problem of labeling a *rectilinear map* with sliding labels. The problem is defined as follows: Given a positive real number h and a set of axis-parallel segments $s_1, \ldots, s_n$ that do not intersect except possibly at endpoints, find a height-h rectangle labeling of $s_1, \ldots, s_n$, that is, axes-aligned closed rectangles $R_1, \ldots, R_n$ such that no two rectangle interiors intersect, rectangle R_i contains segment s_i, the edges of R_i parallel to s_i have the same length as s_i, and those perpendicular to s_i have length h. See Fig. 9 for a rectilinear map with sliding labels.

The link between the two problems is obvious: in both cases the solution consists of positioning axes-aligned objects of fixed size that can slide in one axis direction. In both problems the sliding objects must contain some other given geometric object (a segment or a kernel). Contrary to the segment-labeling problem, in our case the kernels expand in both dimensions and the sliding objects are not all of the same fixed height h, which makes the problem harder. Another, rather technical, difference is that according to the definition of the geometric matching problem, the geometric objects cannot have boundary points in common. This technical detail can be handled by enlarging the objects by an infinitesimal amount, that is, by adding a symbolic ε-distance in the corresponding calculations. In the sequel we will neglect this technical detail. Kim et al. [14] showed that the segment-labeling problem can be solved in $O(n \log^2 n)$ time. We now show how to solve the matching problem in $O(n^2 \log n)$ time.

Note that there is no square matching if two kernels intersect since a kernel is contained in *any* square that matches the corresponding two points. This can be checked in $O(n \log n)$ time by a simple plane sweep [18]. Now recall that α_i is the edge length of Q_i. Whenever the sliding space S_i of a square and a kernel K_j intersect, we *truncate* S_i such that no $\alpha_i \times \alpha_i$-square in $S_i \cup K_i$ intersects the interior of K_j. This can be done via a vertical decomposition in $O(n \log n)$ time. We stop and output "no" if in this process for any i the region $S_i \cup K_i$ does not accommodate an $\alpha_i \times \alpha_i$-square any more.

We define the *interaction graph* $G(\{1, \ldots, n\}, E)$ in which $\{i, j\} \in E$ if and only if the truncated sliding spaces S_i and S_j *interact*, that is, if $S_i \cap S_j \neq \emptyset$.

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

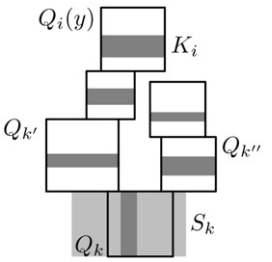


Fig. 10. Edge (k, k') causes $Q_i(y)$, but (k, k'') does not.

Algorithm 1. ComputePositions($Q_1, \ldots, Q_n$)

```
for i ← 1 to n do
step 1   if Q_i is vertically sliding then
           Π_i ← Π_i ∪ {(y_i − α_i + β_i, (i, 0))}        [lower extremal position]
step 2   foreach e ∈ E do
           t_e ← −∞                                       [initialize auxiliary variables]
step 3   foreach (j, i) ∈ E do
(a)        if Q_j is horizontally sliding then
             Π_i ← Π_i ∪ {(y_j + α_j, (j, i))}            [y-coordinate of top edge]
           else
(b)          foreach (y, e) ∈ Π_j with Q_j(y) ∩ S_i ≠ ∅ do
               t_e ← max{t_e, y + α_j}                    [update position of Q_i caused by e]
step 4   foreach e ∈ E do
           if t_e > −∞ then
             Π_i ← Π_i ∪ {(t_e, e)}
return Π_1, ..., Π_n
```

Lemma 3. *The interaction graph G has linear size.*

Proof. A truncated sliding space S_i intersects only a constant number of truncated sliding spaces S_j with $\alpha_j \geqslant \alpha_i$. This is due to the fact that (a) each S_j with $\alpha_j \geqslant \alpha_i$ that intersects S_i must contain a corner of S_i and (b) each of the four corners of S_i lies in at most one sliding space of a horizontally sliding and a vertically sliding square. Thus $|E| \in O(n)$. □

Let (x_i, y_i) be the lower left corner of kernel K_i. Define the relation $K_i < K_j$ to hold if $y_i < y_j$ or if $y_i = y_j$ and $x_i < x_j$. In the sequel we assume that $K_1 < \cdots < K_n$. Now we direct the edges of the interaction graph G, namely from small to large index (according to the new order). For ease of disposition we add a dummy node 0 and dummy edges $(1, 0), \ldots, (n, 0)$ to G.

Now we discretize the problem. For each point pair $\{p_i, q_i\}$ in M we compute $O(n)$ positions of the minimal square Q_i that contains $\{p_i, q_i\}$. We only detail how to do this for vertically sliding squares, the algorithm for horizontally sliding squares is analogous. We denote the (sliding) square Q_i in position y by $Q_i(y)$. We say that an edge $(k, k') \in E$ *causes* $Q_i(y)$ if

(a) there is a directed path $k = v_1, v_2, \ldots, v_m = i$ in G,
(b) the squares $Q_{v_2}, \ldots, Q_{v_m}$ are vertically sliding,
(c) Q_k is vertically sliding if $k' = 0$, else Q_k is horizontally sliding and $v_2 = k'$, and
(d) $\overline{y_k} + \alpha_{v_2} + \cdots + \alpha_{v_{m-1}} = y$, where $\overline{y_k}$ is the y-coordinate of the top edge of K_k.

See Fig. 10 for illustration.

For $i \in \{1, \ldots, n\}$ our algorithm ComputePositions (see Algorithm 1) computes a set Π_i of pairs of the form (y, e), where $y \in \mathbb{R}$ is the y-coordinate of some position of Q_i and $e \in E$ causes $Q_i(y)$. The algorithm assumes the above order of the kernels. An analogous method can be used to calculate sets Π_i for horizontally sliding squares.

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

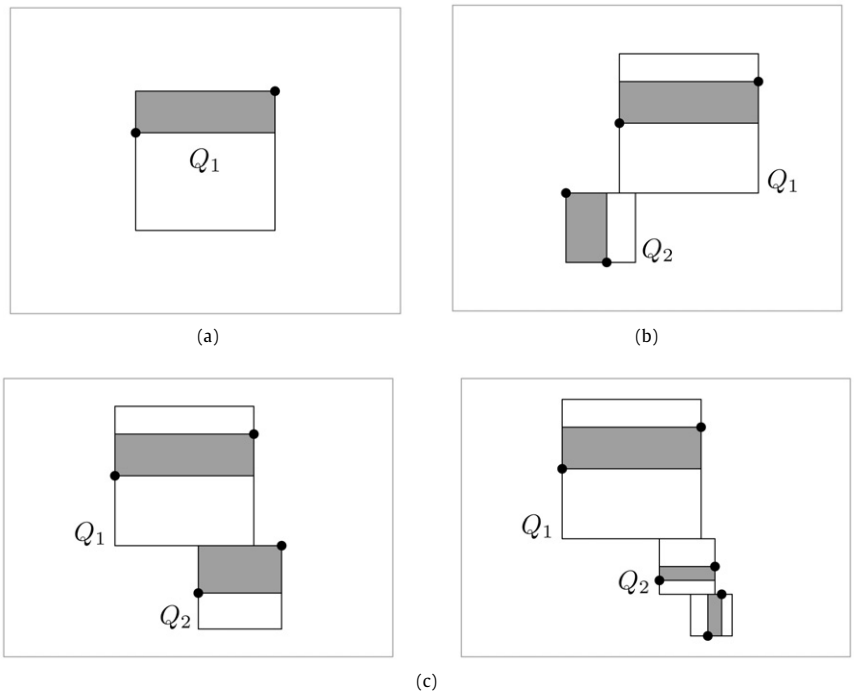


Fig. 11. Reasons why a vertically sliding square Q_1 cannot be moved downwards any further during canonization. (a) Q_1 is in an extremal position. (b) Q_1 hits the kernel or the sliding space of a horizontally sliding square. (c) Q_1 hits a vertically sliding square Q_2 that cannot slide downwards any further.

The asymptotic running time of Algorithm 1 is dominated by the total time spent in step 3(b), which sums up to $O(\sum_{(i,j)\in E} |\Pi_j|)$. Note that for every edge in E there is at most one element in Π_j. Thus $|\Pi_j| \leqslant |E|$, and due to Lemma 3 the algorithm runs in $O(|E|^2) = O(n^2)$ time.

Now assume that there is a strong realization R of the given matching M. We show that we can transform it into a strong realization R' in *canonical form* such that for each square $Q_i(y)$ there is a pair $(y, e) \in \Pi_i$. We go through the squares in order $Q_1, \ldots, Q_n$. Let Q_i be a vertically sliding square. The proof for horizontally sliding squares is analogous. Move Q_i downwards until Q_i reaches its lower extremal position, or the top edge of the sliding space or the kernel of a horizontally sliding square, or the top edge of some other vertically sliding square (that has already been moved). See Fig. 11 for possible positions of Q_i after canonization.

Let $Q_i(y)$ be the resulting position of Q_i. If $Q_i(y)$ is the lower extremal position of Q_i, we are done due to step 1 of our algorithm. If $Q_i(y)$ touches the top edge of a sliding space of a horizontally sliding square, we are done due to step 3(a). Finally, if $Q_i(y)$ touches the top edge of a vertically sliding square $Q_j(z)$ with $z = y - \alpha_j$, then we know (by induction over i) that there is an edge $e \in E$ that has caused $Q_j(z)$ and that fulfills $(z, e) \in \Pi_j$. Due to step 3(b) of the algorithm it is clear that the top edge y of $Q_j(z)$ was considered in the computation of t_e and that $Q_i(y)$ is also caused by $e = (k, k')$. This in turn yields that $(y, e) \in \Pi_i$, since there cannot be another path from k, the origin of e, to i in G that uses only vertically sliding squares and ends in a position with y-coordinate larger than y. Otherwise Q_i would have stopped there during canonization. Thus $Q_i(y) \in \Pi_i$, and we conclude that every strong realization can be transformed into a canonical one.

After Algorithm 1 has computed the sets of type Π_i, it remains to check whether the square positions stored in these sets can be combined such that no two squares overlap. Poon et al. [20] showed that this check reduces to finding a satisfying truth assignment of a 2-SAT formula. They give an $O(k_{\max} n^2)$-time algorithm, where $k_{\max} = \max_i |\Pi_i|$. Strijk and van Kreveld [23] improved the running time to $O(k_{\max} n \log n)$. Since in our case $k_{\max} \in O(n)$, the check takes us $O(n^2 \log n)$ time and thus dominates the running time of Algorithm 1.

Since every strong square matching can be mapped to one in canonical form as described above, the non-satisfiability of the 2-SAT formula implies the non-existence of a strong square matching. On the other hand, if the 2-SAT formula is satisfiable, the corresponding truth assignment translates into a strong square matching (in canonical form). We conclude with the following theorem.

Theorem 4. *Given a set P of n points and a combinatorial matching $M \subseteq \binom{P}{2}$, it can be decided in $O(n^2 \log n)$ time whether M has a strong square realization.*

5. Application to point labeling

In this section we show that the algorithm for strong square matching described in Section 4 can be applied to solve a new map-labeling problem, where points are to be labeled with sliding labels. Recall that in the usual models for point-

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

labeling, a label has a set of anchor points on its boundary. The idea behind the anchor points is that the label must touch the point it labels in one of the anchor points. Further recall that we say that a label is sliding if its anchor points form anchor segments.

In our discussion here we make use of the slider models 1SH, 1SV, and 4S of Poon et al. [19], which define the anchor segments of a label to be its bottom edge, its left edge, and its entire boundary, respectively. We show that our algorithm for strong square matching can be applied to instances of the new model 1SH + 1SV, where for each input point the anchor segment is an arbitrary edge of the label.

This model generalizes the models 1SH and 1SV. For the model 1SH and unit-height rectangular labels, van Kreveld et al. [24] gave an $O(n \log n)$-time factor-2 approximation for *label-number maximization*, that is, an algorithm that labels at least $n_{\max}/2$ points, where $n_{\max}$ is the maximum number of points that can be labeled. The complexity status of the decision problem was unknown. Our algorithm below shows that one can efficiently decide whether all points can be labeled. Van Kreveld et al. showed that for the labeling model 4S the decision problem is NP-hard, even for unit-square labels.

Formally, an instance of 1SH + 1SV is a quadruplet $I = (P, w, h, \chi)$, where $P = \{p_1, \ldots, p_n\}$ is a set of n points in the plane, $w, h : \{1, \ldots, n\} \to \mathbb{R}_{>0}$ are *label-size functions*, and $\chi : \{1, \ldots, n\} \to \{\text{left, right, top, bottom}\}$ is the *anchor function*. We want to determine whether I is *feasible*, that is, whether there are pairwise disjoint closed axes-aligned rectangles $L_1, \ldots, L_n$ in the plane such that for $i = 1, \ldots, n$ the rectangle L_i has width $w(i)$, has height $h(i)$, and touches p_i with its $\chi(i)$ edge. We call such a set of labels *legal*.

We first treat the case of square labels.

Lemma 4. *Given an instance $I = (P, h, h, \chi)$ of the* 1SH + 1SV *labeling problem with square labels, we can decide in $O(n^2 \log n)$ time whether I is feasible. If so, we can compute a legal set of labels within the same time bound.*

Proof. The transformation of I to an instance of the square-matching problem is obvious. For example, if $\chi(i) =$ bottom, define the point $p_i' = (x_i, y_i + h(i))$, where $(x_i, y_i) = p_i$ and $h(i) \times h(i)$ is the size of the label of p_i. Then, apply the algorithm for strong square matching described in Section 4 to the point set $P \cup \{p_i' \mid p_i \in P\}$ and the combinatorial matching $\{\{p_i, p_i'\} \mid p_i \in P\}$. This ensures that each point will be labeled by a label that touches it with the correct edge and that no two labels intersect. □

We now consider the corresponding label-size maximization problem for the special case that all labels are axes-aligned squares of the same size. Note that the case of axes-aligned congruent rectangular labels can be reduced to the square case by scaling the given instance in one coordinate direction.

Theorem 5. *Given a set P of n points in the plane and an anchor function χ, we can compute, in $O(n^2 \log^2 n)$ time, the largest real $\sigma_{\max}$ such that the instance $(P, \sigma_{\max}, \sigma_{\max}, \chi)$ of the 1SH + 1SV labeling problem with congruent square labels is feasible.*

Proof. Let D be a list of all pairwise L_∞-distances in the given point set P and let $D' = \{d/i \mid d \in D, i \in \{1, \ldots, n\}\}$. Note that D' contains $\sigma_{\max}$, since in a solution with labels of maximum size, a subset of the labels, all sliding into the same direction (say, horizontally), must be tightly stacked between two labels that are sliding in the other direction (say, vertically) or in opposite extreme positions. Thus the length or height of the stack is the L_∞-distance between two points in P, and $\sigma_{\max}$ is that distance divided by the number of labels in the stack.

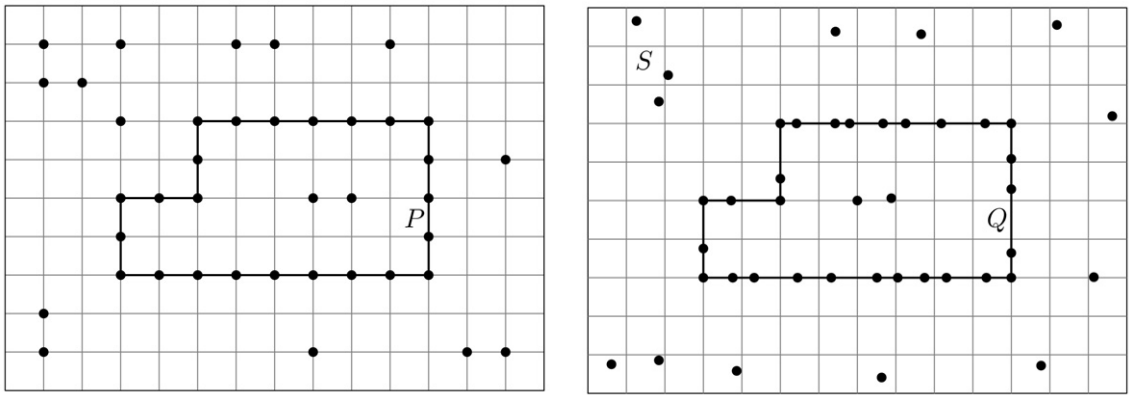
It is clear that D' has cubic size. We can compute and sort D' in $O(n^3 \log n)$ total time. Thus we can do binary search in D', calling the decision algorithm of Lemma 4 in each step of the search, which takes $O(n^2 \log n)$ time per call. After $O(\log n)$ steps we have found $\sigma_{\max}$. This approach yields a total running time of $O(n^3 \log n)$.

We can, however, do better. Instead of sorting D' explicitly, we use the same technique as Duncan et al. [7] for solving the three-label point labeling problem, where a set of n point is to be labeled with $3n$ axes-aligned square labels, three per point, such that each label touches its respective point, no two labels intersect, and the common size of the labels is maximized. Their optimal label size is contained in the same set D' as ours. Their technique, which we sketch below, is reminiscent of the algorithm of Blum et al. [3] for finding a median in linear time.

As in a usual binary search, Duncan et al. maintain an interval $[b, t)$ that contains $\sigma_{\max}$ and shrinks in each step of the search. In addition, they maintain for each point-to-point distance $d \in D$ a pair (x_d, w_d), where x_d is the median of $\mathcal{L}_d = \{d/i \mid i = 1, \ldots, n\} \cap [b, t)$ and w_d is the cardinality of $\mathcal{L}_d$. Note that $\mathcal{L} = \bigcup_{d \in D} \mathcal{L}_d$ contains $\sigma_{\max}$. Instead of replacing one of b or t by the median of $D' \cap [b, t)$ as in usual binary search, Duncan et al. use the weighted median of $\mathcal{L}$, where the values of type w_d are the weights of the values of type x_d. This weighted median can be determined in $O(|\mathcal{L}|) = O(|D|) = O(n^2)$ time. Duncan et al. show that the cardinality of $\mathcal{L}$ decreases in each step of their search by a constant factor, which yields that their search needs $O(\log n)$ steps, too. Thus their approach yields a running time of $O(\max\{|D|, T_{\text{decision}}(n)\} \cdot \log n)$, where $T_{\text{decision}}(n)$ is the time complexity of the corresponding decision problem. According to Lemma 4, we have $T_{\text{decision}}(n) = O(n^2 \log n)$. This yields the desired time bound for our label-size optimization algorithm. □

Next we show how to generalize the algorithm for the decision problem in the proof of Lemma 4 to arbitrary axes-aligned rectangular labels.

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001



(a) (b)

Fig. 12. Examples of rectilinear grid polygons. (a) polygon P is odd. (b) polygon Q is odd with respect to the set Γ of points on its boundary, and Q is reconstructable from S.

Theorem 6. *Given an instance $I = (P, w, h, \chi)$ of the $1SH+1SV$ labeling problem with rectangular labels, we can decide in $O(n^3 \log n)$ time whether I is feasible.*

Proof. Obviously we cannot reduce labeling points with rectangles to realizing a combinatorial matching with disjoint *squares*. The correctness proof of the algorithm for strong square matching, however, does not use the fact that points are matched by squares rather than general rectangles. Other than the correctness, the running time of our algorithm is affected; it increases by a linear factor since the maximum number of edges in the (rectangle) interaction graph is quadratic. □

Finally we make a few remarks regarding generalizations of our new labeling model that are of interest in practice. For example, we can decrease the size of the sliding space by using a different auxiliary point. This is helpful if there are physical landmarks on the map—like rivers—that must not be occluded. The label positions of some points can even be fixed by placing the two points to be matched on opposite corners of the label. This is useful in a (semi-) dynamic scenario when new points are added to the map and one wants to avoid that too many existing labels change their position.

6. NP-Completeness

In this section we investigate the complexity of strong square matching. Our hardness proof is based on a special kind of polygon that we now introduce. Refer to Fig. 12(a) for an example. All our polygons are simple.

Definition 3. Let P be a polygon whose vertices lie on the grid $\mathbb{Z}^2$ and whose edges are axis-parallel. Then we say that P is a *rectilinear grid polygon*. If additionally every edge of P contains an odd number of grid points (that is, has even length), we say that P is *odd*.

Lemma 5. *Let P be an odd polygon and let $\Gamma = \{p_0, \ldots, p_{k-1}\}$ be the grid points on the boundary of P in clockwise order. Suppose that S is a finite set of points such that*

(S1) $\Gamma \subseteq S$,
(S2) *the L_1-distance between any two points $p \in \Gamma$ and $q \in S \setminus \Gamma$ is at least 2.*

If there exists a perfect square matching of S, then every point p_i of Γ is matched to either p_{i-1} or p_{i+1} (modulo k). There are two different perfect square matchings of Γ.

Proof. Let p_i be a point of Γ that is matched to a point $q \in S \setminus \{p_{i-1}, p_{i+1}\}$. Suppose that p_i is not a vertex of P. Then the L_1-distance between p_i and q is at least two. However, if p_i is not a vertex of P, then any square containing both p_i and q contains also p_{i-1} or p_{i+1}, a contradiction. Now if p_i is a vertex of P, then $q \in S \setminus \Gamma$. Since the number of points of Γ is even, there is at least one more vertex $p_j \neq p_i$ of P that is matched to a point $q' \in S \setminus \{p_{j-1}, p_{j+1}\}$. Without loss of generality, we can assume $i < j$ and that every point p_s with $i < s < j$ is matched to either p_{s-1} or p_{s+1}. Since p_i and p_j are vertices of P, $j - i$ is even and the number of points p_s with $i < s < j$ is odd. Thus the points p_s with $i < s < j$ cannot form a perfect matching. This contradiction proves the lemma. □

Now let us slightly relax the requirements for an odd polygon. Refer to Fig. 12 for an example.

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

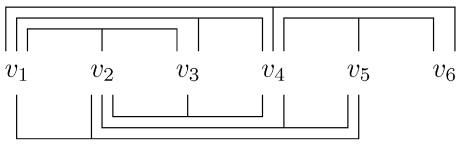


Fig. 13. Embedding of a planar 3-SAT formula.

Definition 4. Let P be an arbitrary rectilinear grid polygon, and let $\Gamma \subset \mathbb{R}^2$ contain all vertices of P and a finite number of other points on the boundary of P. Let the points $\{p_0, \ldots, p_{k-1}\}$ in Γ be sorted in clockwise order around P. If every edge of P contains an odd number of points in Γ, we say that P is *odd with respect to* Γ.

Let $S \supseteq \Gamma$. If any non-vertex point $p_i \in \Gamma$ can be square-matched to p_{i-1} and p_{i+1} (modulo k) but to no other point in S, then we say that P is *reconstructable* from S.

Note that Definition 4 indeed generalizes Definition 3 in the following sense: if a rectilinear grid polygon P is odd and Γ is the set of grid points on its boundary, then P is both odd with respect to Γ and reconstructable from Γ. This allows us to generalize Lemma 5 as follows.

Corollary 2. *Let P be a rectilinear grid polygon, and let S be a finite set of points such that P is reconstructable from S and such that P is odd with respect to the set Γ of points in S that lie on the boundary of P. Suppose the points $\{p_0, \ldots, p_{k-1}\}$ in Γ are sorted in clockwise order around P.*

If there exists a perfect square matching of S, then every point p_i of Γ is matched to either p_{i-1} or p_{i+1} (modulo k). There are two different perfect square matchings of Γ.

The proof of Corollary 2 is analogous to that of Lemma 5. Now we can prove the main theorem of this section.

Theorem 7. *It is* NP*-hard to decide whether a given point set admits a perfect strong square matching.*

Proof. Our proof is by reduction from Planar3Sat, which is NP-hard [15]. Let φ be a planar 3-SAT formula. Note that the variables and clauses of φ can be embedded in the plane as in Fig. 13 where all variables lie on a horizontal line and all clauses are represented by *non-intersecting* three-legged combs [13]. Based on this embedding we construct a finite point set S such that S has a perfect strong square matching if and only if φ is satisfiable.

For an overview of our construction, see Fig. 14. The construction has three main building blocks: variable gadgets (dark shaded boxes in Fig. 14), clause gadgets (light shaded comb in Fig. 14), and adapters that mediate between variable and clause gadgets.

An important point configuration that is used in several places of our construction is what we call a *stopper*, that is, a set of eight points arranged in a tiny 3×3 grid without the center point. In Fig. 14 all occurrences of stoppers are marked by little crosses ($\times$). Since a stopper is a miniature of an odd polygon, Lemma 5 guarantees that its points can only be matched to neighboring points on its boundary, but not to any other points in S. We use stoppers to exclude certain point pairs from a matching or to force squares in a matching into certain positions.

We first describe our variable gadgets. We refer to the dark-shaded boxes in Fig. 14 as *variable boxes*. They all have height 2; the length of a variable box is an odd integer and roughly proportional to the number of clauses in which the corresponding variable occurs. Note that a variable box B is odd with respect to the set Γ_B of points on its boundary. Moreover, B is reconstructable from $S \supseteq \Gamma_B$. Thus Corollary 2 guarantees that the points on the boundary of a variable box can only be matched among each other and only in two different ways. Refer to Fig. 14; in each subfigure the point pairs in the matching are indicated by bold line segments or, in crucial places, by the appropriate squares. If the center point of the left edge of a variable box is matched to its neighbor above, the corresponding variable is set to true, otherwise it is set to false.

Next we turn to the legs of the clause gadgets. We call the lowest point on the right side of a clause leg its *heel point*. For example, the point p in Fig. 14 is the heel point of the rightmost clause leg. Observe that a heel point can only be matched to either its nearest horizontal neighbor or its nearest vertical neighbor. In the latter case we say that the leg *transmits pressure*. This is the case, for example, for the point p in Fig. 14(a), but not for the same point in Fig. 14(b). We use bold vertical arrows in the legs in Fig. 14 to indicate the existence of pressure; arrows pointing upwards indicate that pressure is transmitted, whereas arrows pointing downwards indicate that no pressure is transmitted. Note that our description assumes that the clause gadget lies above the variable boxes; the other case is symmetric.

Now let us describe the adapters. An adapter consists of the bottommost points of a clause leg, of which there are five, and the five points on the boundary of the corresponding variable box that lie in the vertical projection of the clause leg. In the legend in Fig. 14(a) we have labeled these points from top to bottom and from left to right using the numbers $1, \ldots, 10$. The adapters make sure that pressure is transmitted if and only if either (a) the variable (such as v or w in Fig. 14(a)) is set to false and occurs as a positive literal in the clause or (b) the variable (such as u in Fig. 14(a)) is set to true, but occurs as

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

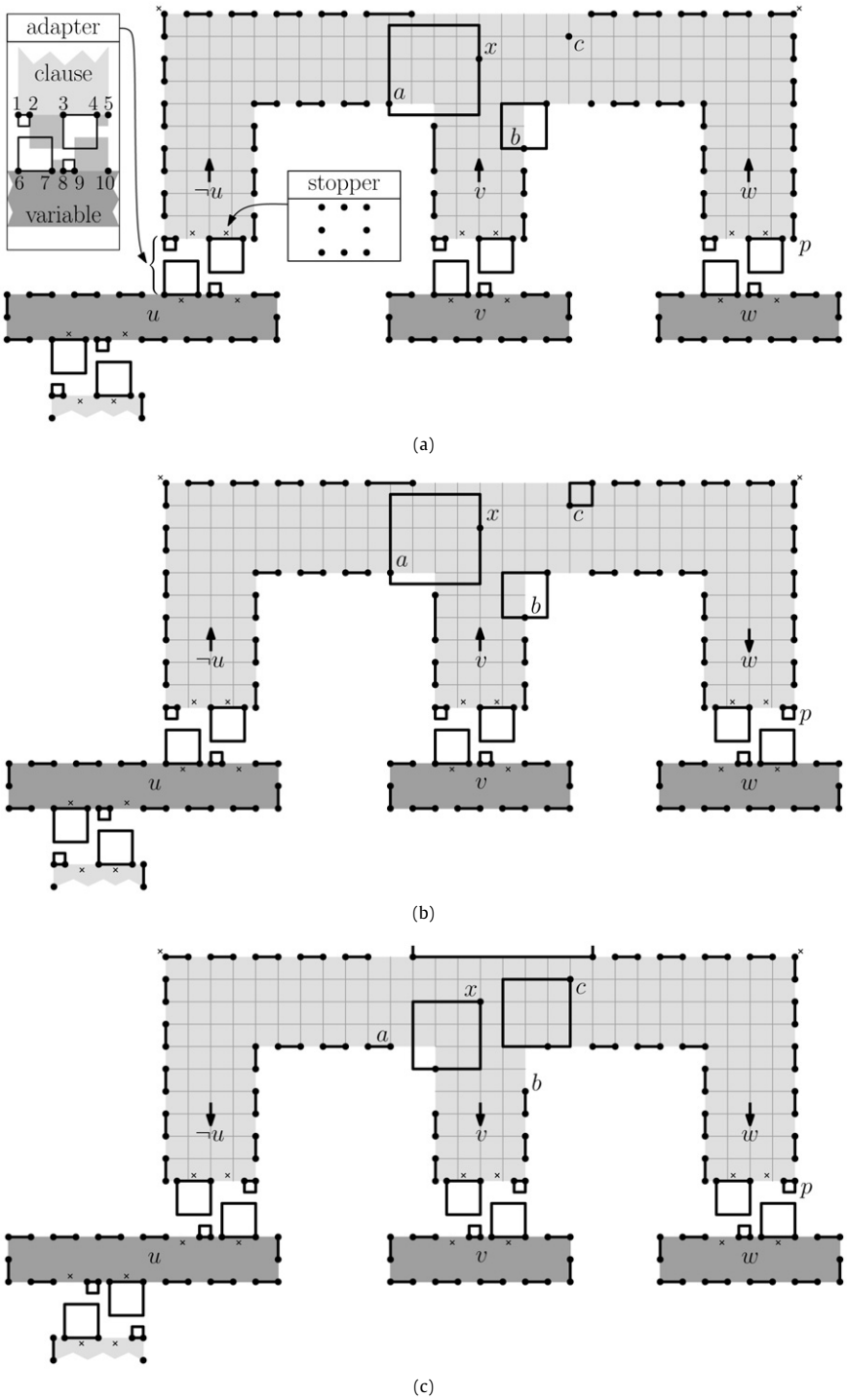


Fig. 14. A gadget for the clause $\neg u \vee v \vee w$. (a) Non-perfect matching corresponding to $u = true$, $v = false$, $w = false$. (b) Perfect matching corresponding to $u = true$, $v = false$, $w = true$. (c) Perfect matching corresponding to $u = false$, $v = true$, $w = true$.

a negated literal. The difference between the two cases is the horizontal distance δ of the leftmost adapter point (point 6) from the left edge of the variable box; in case (a) δ is even, whereas in case (b) δ is odd.

The distance of adapter points that are horizontal neighbors is either 0.5 or 1.5 units. Exactly one pair of adapter points in the variable box must be matched by a large, that is, a 1.5×1.5 square: either points 6 and 7, or points 9 and 10. Note that there are stoppers close to and slightly below the midpoint of each of these point pairs. The stoppers force the large square (nearly) into its topmost position.

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

Similarly, exactly one pair of adapter points in the clause leg must be matched by a large square: point 3 is either matched to point 2 or to point 4. Here, stoppers force the large square (nearly) into its bottommost position. Since the vertical distance between clause and variable gadgets is 2.5, which is less than $2 \cdot 1.5$, only one combination of large squares is possible; either 2–3 and 9–10, or 6–7 and 3–4. Only the latter combination forces point 5 to be matched to its nearest vertical neighbor, which means that the leg transmits pressure. It is not hard to see that a large square that matches points 6 and 7 corresponds to case (a) if δ is even and to case (b) if δ is odd, which is what we desired.

Note that an adapter has width 4 and contains five points. This is exactly the same as the number of grid points on a horizontal line segment of length 4. Thus, an adapter does not "desynchronize" a variable box; the (parity) arguments above apply for *any* adapter on the top edge of a variable box. Due to the stoppers there is no problem if adapters on the top and the bottom edge of a variable box overlap horizontally.

Finally, we discuss the remaining features of our clause gadget. In its central part—where the three clause legs meet—there are four special points a, b, c, and x with the property that x has L_1-distance 4 from each of a, b, and c. The position of the special points is chosen such that x can be matched with each of the other three special points, but no two of those can be matched to each other. The gadget in its entirety is built such that two points cannot be matched if all three legs transmit pressure. For an example, see the point c and another point in its vicinity in Fig. 14(a). This corresponds to the situation where all three literals of a clause are false.

We now argue that no point of a clause gadget can be matched to any point of another clause gadget. The only candidates for points that could possibly be matched are the top corner points. All other points can only be matched by squares up to a fixed constant side length (at most 10), and keeping clause gadgets at a larger distance avoids the problem. To solve the problem for the top corner points we simply place stoppers appropriately, see Fig. 14.

On the other hand we claim that all points in a clause gadget can be matched if at most two legs transmit pressure. To prove the claim it is enough to check all seven cases of at most two legs transmitting pressure. Fig. 14 (b) and (c) depict two of these cases. We conclude that the point set S has a perfect square matching if and only if the planar 3-SAT formula φ is satisfiable. Our reduction is polynomial. □

Corollary 3. *Perfect strong square matching is* NP*-complete.*

Proof. Theorem 7 yields the NP-hardness. To show that the problem actually lies in $\mathcal{NP}$, we non-deterministically guess a combinatorial matching. Then we have to decide deterministically and in polynomial time whether this matching has a strong square realization. For this we use the algorithm of Theorem 4. □

7. Open problems

In this paper we showed that in any set of n points at least $2\lfloor n/3 \rfloor$ can be strongly matched using axes-aligned rectangles. We have a matching lower bound (see Fig. 1). It remains open, however, whether strong maximum-cardinality rectangle matchings for individual point sets can be computed efficiently.

Given a set of points and a combinatorial matching, we can test efficiently whether the matching has a realization as a weak or a strong square matching. Our algorithm for weak square realizations runs in optimal $O(n \log n)$ time. For strong square realizations, however, there is still a linear-size gap between our $O(n \log n)$ lower bound and our $O(n^2 \log n)$-time algorithm.

We have shown that the algorithm for strong square realizations solves the decision version of an interesting new point-labeling problem. We have given an efficient solution of the corresponding label-size maximization problem for the special case of axes-aligned square labels of equal size. Can this be generalized to arbitrary axes-aligned rectangles?

We also proved that perfect strong square matching (without a given combinatorial matching) is NP-complete. We conjecture that the same holds for perfect weak square matching, but the adapters in our proof fail in the weak case. We further conjecture that strong rectangle and disk matching are both NP-hard.

What about geometric matching in 3-space?

Acknowledgements

We thank Marc van Kreveld for information on Ref. [23]. We are indebted to Mikio Kano and Ferran Hurtado for sending us long versions of their article [1]. We thank the anonymous referees for helpful comments. We are grateful to Sándor Fekete for pointing out to us the lower-bound construction in Theorem 3.

References

[1] B.M. Ábrego, E.M. Arkin, S. Fernández-Merchant, F. Hurtado, M. Kano, J.S.B. Mitchell, J. Urrutia, Matching points with circles and squares, in: J. Akiyama, M. Kano, X. Tan (Eds.), Proc. 8th Japanese Conf. Discrete Comput. Geom. (JCDCG'04), in: Lecture Notes Comput. Sci., vol. 3742, Springer-Verlag, 2005, pp. 1–15.

[2] B.M. Ábrego, E.M. Arkin, S. Fernández-Merchant, F. Hurtado, M. Kano, J.S.B. Mitchell, J. Urrutia, Personal communication, April 2008.

[3] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, R.E. Tarjan, Time bounds for selection, J. Comput. Syst. Sci. 7 (4) (1973) 448–461.

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001

[4] M.A. Bekos, M. Kaufmann, A. Symvonis, A. Wolff, Boundary labeling: Models and efficient algorithms for rectangular maps, Comput. Geom. Theory Appl. 36 (3) (2007) 215–236.
[5] M. Ben-Or, Lower bounds for algebraic computation trees, in: Proc. 15th Annual ACM Sympos. Theory Comput. (STOC'83), 1983, pp. 80–86.
[6] M.B. Dillencourt, Toughness and Delaunay triangulations, Discrete Comput. Geom. 5 (1990) 575–601.
[7] R. Duncan, J. Qian, A. Vigneron, B. Zhu, Polynomial time algorithms for three-label point labeling, Theoret. Comput. Sci. 296 (1) (2003) 75–87.
[8] A. Dumitrescu, W. Steiger, On a matching problem in the plane, Discrete Math. 211 (2000) 183–195.
[9] T. Erlebach, T. Hagerup, K. Jansen, M. Minzlaff, A. Wolff, Trimming of graphs, with an application to point labeling, in: S. Albers, P. Weil (Eds.), Proc. 25th Internat. Sympos. Theoretical Aspects Comput. Sci. (STACS'08), Bordeaux, 2008, pp. 265–276.
[10] M. Formann, F. Wagner, A packing problem with applications to lettering of maps, in: Proc. 7th Annu. ACM Sympos. Comput. Geom. (SoCG'91), 1991, pp. 281–288.
[11] M. Jiang, S. Bereg, Z. Qin, B. Zhu, New bounds on map labeling with circular labels, in: R. Fleischer, G. Trippen (Eds.), Proc. 15th Annu. Internat. Sympos. Algorithms Comput. (ISAAC'04), in: Lecture Notes Comput. Sci., vol. 3341, Springer-Verlag, 2004, pp. 606–617.
[12] J.-W. Jung, K.-Y. Chwa, Labeling points with given rectangles, Inform. Process. Lett. 89 (3) (2004) 115–121.
[13] D.E. Knuth, A. Raghunathan, The problem of compatible representatives, SIAM J. Discrete Math. 5 (3) (1992) 422–427.
[14] S.K. Kim, C.-S. Shin, T.-C. Yang, Labeling a rectilinear map with sliding labels, Internat. J. Comput. Geom. Appl. 11 (2) (2001) 167–179.
[15] D. Lichtenstein, Planar formulae and their uses, SIAM J. Comput. 11 (2) (1982) 329–343.
[16] E.M. McCreight, Priority search trees, SIAM J. Comput. 14 (2) (1985) 257–276.
[17] S. Micali, V.V. Vazirani, An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs, in: Proc. 21st IEEE Symp. Found. Comp. Sci. (FOCS'80), 1980, pp. 17–27.
[18] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, third ed., Springer-Verlag, New York, 1990.
[19] S.-H. Poon, C.-S. Shin, T. Strijk, T. Uno, A. Wolff, Labeling points with weights, Algorithmica 38 (2) (2003) 341–362.
[20] C.K. Poon, B. Zhu, F. Chin, A polynomial time solution for labeling a rectilinear map, Inform. Process. Lett. 65 (4) (1998) 201–207.
[21] Z. Qin, A. Wolff, Y. Xu, B. Zhu, New algorithms for two-label point labeling, in: M. Paterson (Ed.), Proc. 8th Annual European Sympos. Algorithms (ESA'00), in: Lecture Notes Comput. Sci., vol. 1879, Springer-Verlag, 2000, pp. 368–379.
[22] F. Rendl, G. Woeginger, Reconstructing sets of orthogonal line segments in the plane, Discrete Math. 119 (1993) 167–174.
[23] T. Strijk, M. van Kreveld, Labeling a rectilinear map more efficiently, Inform. Process. Lett. 69 (1) (1999) 25–30.
[24] M. van Kreveld, T. Strijk, A. Wolff, Point labeling with sliding labels, Comput. Geom. Theory Appl. 13 (1999) 21–47.
[25] A. Wolff, T. Strijk, The map-labeling bibliography, http://i11www.ira.uka.de/map-labeling/bibliography, 1996.
[26] A.C.-C. Yao, Lower bounds for algebraic computation trees of functions with finite domains, SIAM J. Comput. 20 (4) (1991) 655–668.

Please cite this article in press as: S. Bereg et al., Matching points with rectangles and squares, Computational Geometry (2008), doi:10.1016/j.comgeo.2008.05.001