# Matching Points with Rectangles and Squares[⋆]

Sergey Bereg[1], Nikolaus Mutsanas[2], and Alexander Wolff[2]

[1] Department of Computer Science, University of Texas at Dallas, U.S.A.
Email: besp@utdallas.edu

[2] Fakultät für Informatik, Universität Karlsruhe, P.O. Box 6980, D-76128 Karlsruhe, Germany. Email: Nikolaus.Mutsanas@stud.uni-karlsruhe.de,
WWW: http://i11www.ira.uka.de/people/awolff

**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 point set $P$, 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 axis-aligned squares and rectangles. We give algorithms for these classes and show that it is NP-hard to decide whether a point set has a perfect strong square matching. We show that one of our matching algorithms solves a family of map-labeling problems.

## 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 segments to match points, a matching can be defined to consist of axis-aligned rectangles that contain exactly two points. Analogously, a matching can be defined to consist of elements of any family of convex geometric objects, like squares and disks. This class of geometric matching problems has been introduced by Ábrego et al. [1].

In this paper we deal with the problem of matching points with axis-aligned rectangles and squares. Given a set of points in the plane, a rectangle matching is a set of axis-aligned closed rectangles such that each rectangle contains exactly two points of the point set. A square matching is defined analogously for axis-aligned squares. A geometric matching of either type is called *strong*, if the geometric objects do not intersect. Otherwise the matching is called *weak*. 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.

Whereas a strong perfect rectangle matching always exists if the point set is in general position, we show in Section 3 that there are point sets which only allow strong rectangle matchings that cover less than a fraction of 2/3 of the points. We also give an algorithm that always matches $4n/7 - 5$ out of $n$ points.

[⋆] Work supported by grant WO 758/4-2 of the German Science Foundation (DFG).

For the problem of matching points with axis-aligned squares, we give in Section 4 efficient algorithms that decide the following problems: 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? In Section 5 we show that our algorithm for strong square matching can also be used to solve a family of point-labeling problems. Finally, in Section 6 we show that it is NP-hard to decide whether a given point set admits a perfect strong square matching.

## 2 The general problem

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

**Definition 1.** *Let $\mathcal{C}$ be a family of geometric objects and $P$ a point set with an even number of points. A $\mathcal{C}$-matching for $P$ 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, and it is called* perfect *if every $p \in P$ is contained in some $C \in M$.*

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 $\mathcal{C}, P$ 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 those two points, i.e. $C \cap P = \{p, q\}$. We regard a geometric matching as a* realization *of the underlying combinatorial matching.*

The problem of matching points with geometric objects was introduced by Ábrego et al. [1]. Their results base on the assumption that the points are in general position, i.e. if there are no two points with the same $x$- or $y$-coordinate. Ábrego et al. showed that for a point set in general position a matching with axis-aligned squares always exists. They also showed that for every point set $P$ with $n$ points, there is always 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, they give point sets with $13m$ points such that any strong matching covers at most $6m$ of the points.

For the family $\mathcal{D}$ of disks, Ábrego et al. also showed that a disk matching always exists, provided that $n$ is even. They prove this assumption by using the matching graph for $\mathcal{D}$. 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 [2] proved that for $n$ even the Delaunay triangulation always contains a perfect matching. Ábrego et al. also showed that for any point set there is always a strong disk matching covering at least $2\lceil (n-1)/8 \rceil$ points. On the other hand, there are arbitrarily large point sets with $n$ points, such that any strong disk matching covers at most $72n/73$ points.

## 3 Rectangle matching

If points are in general position, the problem of matching points with axis-aligned rectangles is trivial. An obvious algorithm that yields a perfect strong rectangle matching is the following: Sort the points lexicographically and connect every point with odd index to its successor. Since the order of the $x$-coordinates is strictly monotonous, two consecutive rectangles cannot overlap.

However, if we drop the condition of general position, the problem becomes interesting. Consider the point set $P_n = \{(i, i), (i-1, i), (i, i-1) \mid 1 \leq i \leq n\} \cup \{(n, n+1), (n+1, n)\}$ and its matching graph $G_n$, which has $3n+2$ vertices and $4n$ edges. Each of the *central* nodes $(1, 1), \ldots, (n, n)$ has degree 4 in $G_n$, 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 $P_n$ has cardinality $n$; it covers $2n$ out of $3n+2$ points. 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 a simple and efficient algorithm that always yields a strong rectangle matching that covers at least $4n/7 - c$ points in an $n$-element point set, where $c$ is a small constant. The algorithm has been implemented and is accessible via a Java applet at the URL `http://i11www.ira.uka.de/matching`. The idea of our algorithm is the following. We partition the given point set $P$ into *vertical subsets* $V_i$, maximum chains of points with equal $x$-coordinate (allowing $|V_i| = 1$). Let $v_i$ be the cardinality of $V_i$. We process these subsets from left to right, making a cut as soon as a matching for the current point set has been found that matches at least a fraction of $4/7$ of the points since the last cut. After making a cut, we disregard the points already processed and start over again. If each of the subsets—except possibly the last—has a matching that covers at least a fraction of $4/7$ of the points, the overall matching will cover at least $4n/7 - c$ of the points in $P$ if $c$ is the size of the last subset.

If $v_1$ is even or if $v_1 \geq 3$ is odd, we can match at least a fraction of $2/3 > 4/7$ of the points and make a cut after $V_1$. Thus we assume $v_1 = 1$ and consider $V_1 \cup V_2$. Again, if $v_2$ is even or if $v_2 \neq 3$ is odd, we can match enough points to make a cut after $V_2$. However, if $v_2 = 3$, the middle point in $V_2$ may have the same $y$-coordinate as the point in $V_1$, see Figure 1 (a). This is the only configuration with cardinalities $v_1 = 1$ and $v_2 = 3$ (for short $[1, 3]$) that we cannot match perfectly. In this case consider the point set $V_1 \cup V_2 \cup V_3$. If $v_3$ is even or if $v_3 \geq 5$ is odd, we can match enough points to make a cut after $V_3$. The cases $[1, 3, 1]$ and $[1, 3, 3]$ need to be considered separately.

In case $v_3 = 1$, the points may only allow two out of five points to be matched, as shown in Figure 1 (b) (in every other configuration we can match four out of five points and make a cut). In this case, consider $V_1 \cup V_2 \cup V_3 \cup V_4$. It can be shown that we can always match at least $2/3$ of these points, regardless of the cardinality of $V_4$ (see Figures 1 (c) and (d)).

In case $v_3 = 3$, we can always match four points within their respective vertical sets. This covers four out of seven points. and allows us to make a cut after $V_3$.

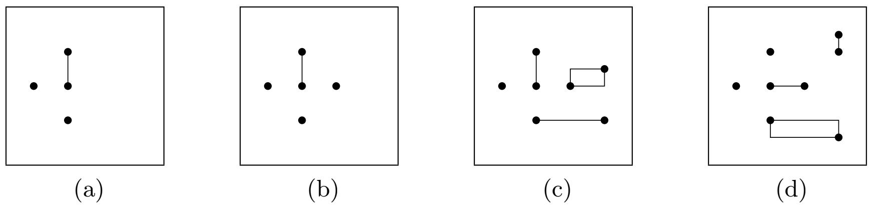


**Fig. 1.** Some special cases for the 4/7-approximation: a) only 2 out of 4 points can be matched, b) only 2 out of 5 points can be matched. Matching that covers at least 2/3 of the points c) for $v_4$ even and d) for $v_4$ odd.

Note that there might be some points left after the last cut, that cannot be processed. The number of left-over points is at most 5, since we can always cut after $[1, 3, 1, *]$ and after $[1, 3, \geq 2]$. This means that our algorithm in fact always matches at least $4n/7 - 5$ out of $n$ points. We assume that a similar technique may be used to find an approximation algorithm that covers at least 2/3 of the points minus some constant, which would reach the highest fixed ratio that can be guaranteed. The time complexity of our algorithm is $O(n \log n)$ due to the lexicographical sorting. We summarize:

**Theorem 1.** *There are point sets of arbitrary size where any strong rectangle matching covers at most 2/3 of the points. There is an algorithm that computes in $O(n \log n)$ time a strong rectangle matching that covers at least $4n/7-5$ points in an $n$-element point set.*

## 4 Square matching

Note that, contrary to rectangle matchings, a square matching is not uniquely defined by a given 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 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.

How many ways are there to place a minimum-size square $Q_i$ that contains two given points $p_i$ and $q_i$? It is easy to see that the edge length $\alpha_i$ of a minimal square is the distance of the two points in the maximum (or $L_\infty$-) metric (see Figure 2). If the two coordinate differences are not equal, the square can be slid—to some extend—in the direction of the smaller coordinate difference $\beta_i$. This leads to the model of the sliding squares illustrated in Figure 2.

The *kernel* $K_i$ of a point pair $\{p_i, q_i\}$ is their bounding box, i.e. the smallest axis-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 Figure 2. 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.*

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 $L_\infty$-distance $a$ and $b$ is larger than the $L_\infty$-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 $L_\infty$-distance of $a$ and $b$ is at most that of $p$ and $q$, $M$ does not have a square realization. Using priority search trees [6], 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.*

Now we turn to the problem of finding a strong square realization for a given combinatorial matching. Due to the above observations, this problem simplifies 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 we considered.

It turns out that a problem in map labeling is related to our problem, namely the problem of labeling a rectilinear map with sliding labels. That problem is defined as follows: Given a set of axis-aligned segments that do not intersect and a positive real $h$, is there a labeling with axis-aligned rectangles with width the same as the segment and height $h$? Each label must contain the segment it labels. See Figure 4 for an example.

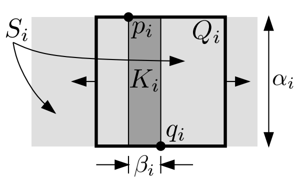


**Fig. 2.** Notation for sliding squares.

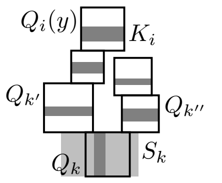


**Fig. 3.** $(k, k')$ causes $Q_i(y)$; $(k, k'')$ doesn't.

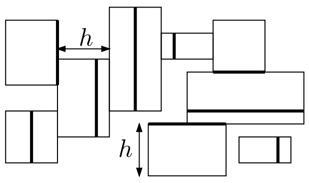


**Fig. 4.** Set of line segments (bold) with height-$h$ rectangle labeling.

The link between the two problems is obvious: in both cases the solution consists of positioning axis-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 map-labeling problem, in our case the kernels expand in both dimensions and the sliding objects are not of fixed height $h$, which makes the problem harder. Kim et al. [3] showed that the map-labeling problem can be solved in $O(n \log^2 n)$ time. In this paper we adapt their algorithm to solve the matching problem within the same time.

Note that there is no strong matching if two kernels intersect. This can be checked in $O(n \log n)$ time by a simple plane sweep [9]. Furthermore, if the sliding space of a square and a kernel intersect, the sliding space can be truncated. This can be done via a trapezoidal (i.e. vertical) decomposition in $O(n \log n)$ time.

We define the *interaction graph* $G(\{1, \dots, n\}, E)$ in which $\{i, j\} \in E$ if and only if the truncated sliding spaces $S_i$ and $S_j$ *interact*, i.e. $S_i \cap S_j \neq \emptyset$. Recall that $\alpha_i$ is the edge length of $Q_i$. It is easy to see that $S_i$ intersects only a constant number of truncated sliding spaces $S_j$ with $\alpha_j \geq \alpha_i$. Thus $|E| \in O(n)$. Let $(x_i, y_i)$ be the lower left corner of kernel $K_i$. Define $K_i < K_j$ to hold if $y_i < y_j$ or if $y_i = y_j$ and $x_i < x_j$. In the sequel we assume that $K_1 < \dots < K_n$. 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 $(1, 0), \dots, (n, 0)$ 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 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 $k = v_1, v_2, \dots, v_m = i$ in $G$, (b) the squares $Q_{v_2}, \dots, Q_{v_m}$ are vertically sliding, (c) $Q_k$ is vertically sliding if $k' = 0$, else $Q_k$ is horizontally sliding and $v_2 = k'$, and (d) $\overline{y_k} + \alpha_{v_2} + \dots + \alpha_{v_{m-1}} = y$, where $\overline{y_k}$ is the $y$-coordinate of the top edge of $K_k$. See Figure 3 for illustration.

For $i \in \{1, \dots, n\}$ our algorithm computes a set $\Pi_i$ 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 $Q_i(y)$.

```
for $i \leftarrow 1$ to $n$ do
 1. if $Q_i$ is vertically sliding then
        $\Pi_i \leftarrow \Pi_i \cup \{(y_i - \alpha_i + \beta_i,\ (i, 0))\}$                {lower extremal position}
```

```
2. for each $e \in E$ do $t_e \leftarrow -\infty$        {initialize auxiliary variables}
3. for each $(j, i) \in E$ do
   (a) if $Q_j$ is horizontally sliding then
          $\Pi_i \leftarrow \Pi_i \cup \{(y_j + \alpha_j, (j, i))\}$        {$y$-coordinate of top edge}
   (b) else        {$Q_j$ vertically sliding}
          for each $(y, e) \in \Pi_j$ with $Q_j(y) \cap S_i \neq \emptyset$ do
             $t_e \leftarrow \max\{t_e, y + \alpha_j\}$        {update position of $Q_i$ caused by $e$}
4. for each $e \in E$ do
      if $t_e > -\infty$ then $\Pi_i \leftarrow \Pi_i \cup \{(t_e, e)\}$
```

The asymptotic running time of the above algorithm 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 $\Pi_j$. Thus $|\Pi_j| \leq |E|$, and 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 of a horizontally sliding square, or the top edge of some other vertically sliding square (that has already been moved). 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 $(z, e) \in \Pi_j$. Now 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 $Q_j(y)$ 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 higher $y$-coordinate than $y$. Otherwise $Q_i$ would have stopped there.

After we have computed the sets of type $\Pi_i$, it remains to check whether the square positions stored in these sets can be combined such that no two squares overlap. Poon et al. [7] showed that this can be solved by examining the satisfiability of a 2-SAT formula in $O(k_{\max} n^2)$ time, where $k_{\max} = \max_i |\Pi_i|$. Strijk and van Kreveld [10] improved the running time to $O(k_{\max} n \log n)$. Since $k_{\max} \in O(n)$, the resulting time complexity of our algorithm is $O(n^2 \log n)$.

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, a witness of its satisfiability translates into a strong square matching (in canonical form). We conclude with the following theorem.

**Theorem 3.** *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 family of point-labeling problems.

Poon et al. [8] describe the problem of labeling points with sliding labels as follows: The labels are fixed-size rectangles that touch the point they label with their boundary. Every label may slide along a fixed direction (horizontally or vertically) and may not occlude other points.

The transformation of an instance of the point-labeling problem to an instance of the square-matching problem is obvious, provided that labels are squares. The algorithm of Section 4 can solve a more general problem though. Varying the position of the auxiliary point, one can vary the size of the quadratic label. The space within which the label may slide can be shortened too, by forming a kernel of the respective thickness with the use of a different auxiliary point. Note that the sliding space may even be shortened asymmetrically. Shortening the sliding space of some labels can be of practical interest, e.g. when there are physical landmarks on the map—like rivers—that must not be occluded.

Note that, even though labeling points with rectangles cannot be reduced to examining the realizability of a square matching, the same techniques used for the square-matching algorithm can be applied to sliding rectangles, too. This generalizes the family of solvable point-labeling problems even further.

## 6 NP-Completeness

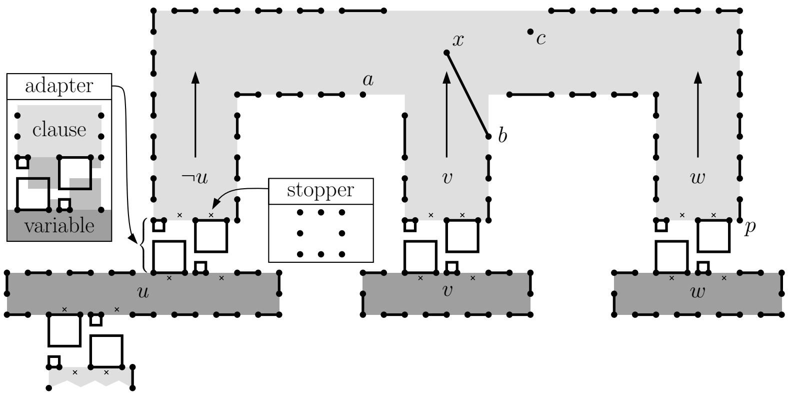
In this section we investigate the complexity of square matching.

**Theorem 4.** *It is NP-hard to decide whether a given point set admits a perfect strong square matching.*

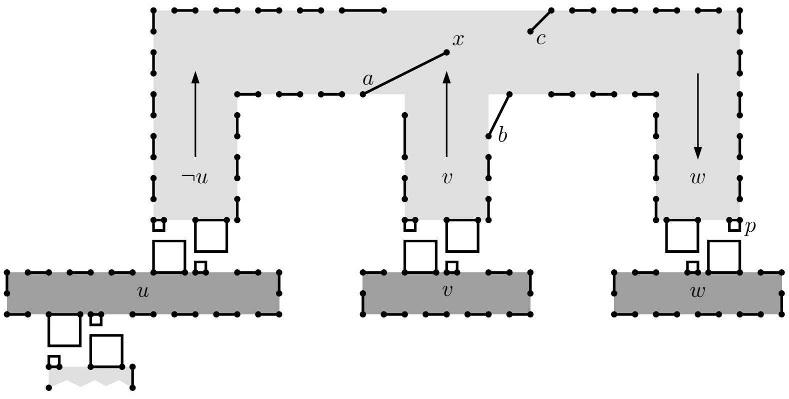
*Proof.* Our proof is by reduction from Planar 3-SAT, which is NP-hard [5]. Note that the variables and clauses of $\phi$ can be embedded in the plane such that all variables lie on a horizontal line and all clauses are represented by *non-intersecting* three-legged combs [4]. Let $\phi$ be a planar 3-SAT formula. We construct a finite point set $S$ such that $S$ has a perfect square matching if and only if $\phi$ is satisfiable.

For the general layout of our variable and clause gadgets, see Figure 5. Each variable of $\phi$ is represented by a *box*, i.e. an axis-parallel rectangle (dark shaded in Figure 5). The points on the boundary of these rectangles can only be matched among each other and only in two different ways. This is true for two reasons: neighboring points on the boundary are closer to each other than to any other point, and between any two corner points there is an odd number of other points.

If the center point of the left edge of the rectangle is matched to its neighbor above, the corresponding variable is true, otherwise it is false. The point pairs in the matching are connected by thick solid line segments in Figure 5, respectively. The variable boxes are connected via *adapters* to vertical *pipes*, the legs of our clause gadgets. We say that a pipe *transmits pressure* if the lowest point on its



(a) Non-perfect matching corresponding to $u = true$, $v = false$, $w = false$.



(b) Perfect matching corresponding to $u = true$, $v = false$, $w = true$.

**Fig. 5.** A gadget for the clause $\neg u \vee v \vee w$.

right side is matched upwards. This is the case e.g. for the point $p$ in Figure 5 (a), but not in Figure 5 (b). Generally the long vertical arrows in the pipes in Figure 5 indicate that (no) pressure is transmitted if they point upwards (downwards). Note that our description assumes that the clause gadget lies above the variable boxes; the reverse case is symmetric.

The adapters between the variable boxes and the pipes make sure that pressure is transmitted if and only if the variable (such as $v$ or $w$ in Figure 5 (a)) is set to false and occurs as a positive literal in the clause or the variable (such as $u$ in Figure 5 (a)) is set to true, but occurs as a negated literal. For the adapters we need a special construct, so-called *stoppers*, i.e. configurations of eight points arranged in a $3 \times 3$ grid without the center point. A stopper is designed such that its points can only be matched to neighboring points on its boundary, but not to any other points—just like a variable box. The stoppers make sure that the large squares stick out sufficiently far from the variable box and the clause legs to synchronize each other.

Our clause gadget (light shaded in Figure 5) with the special points $a$, $b$, $c$, and $x$ is built such that two points cannot be matched if all three pipes transmit pressure, see e.g. the points $a$ and $c$ in Figure 5 (a). This corresponds to the situation where all three literals of a clause are false. Note that no point of a clause gadget can be matched to any point of another clause gadget if the clauses are nested. In the case of neighboring clause gadgets this can simply be avoided by making sure that they have sufficient distance and different height (or by placing stoppers next to the corner vertices).

On the other hand we claim that all points in a clause gadget can be matched if at most two pipes transmit pressure. To prove the claim it is enough to check all seven cases of at most two pipes transmitting pressure. Figure 5 (b) depicts one of these cases. We conclude that the point set $S$ has a perfect square matching if and only if $\phi$ is satisfiable. Our reduction is polynomial. $\square$

**Corollary 1.** *Perfect strong square matching is NP-complete.*

*Proof.* Theorem 4 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 3. $\square$

**Acknowledgments.** We thank one of the anonymous referees for valuable comments and Marc van Kreveld for information on [10].

## References

1. B. M. Ábrego, E. M. Arkin, S. Fernández-Merchant, F. Hurtado, M. Kano, J. S. B. Mitchell, and J. Urrutia. Matching points with geometric objects: Combinatorial results. In J. Akiyama et al., editors, *Proc. 8th Jap. Conf. Discrete Comput. Geometry (JCDCG'04)*, pages 1–15, volume 3742 of *LNCS*. Springer-Verlag, 2005.
2. M. B. Dillencourt. Toughness and Delaunay triangulations. *Discrete Comput. Geom.*, 5:575–601, 1990.
3. S. K. Kim, C.-S. Shin, and T.-C. Yang. Labeling a rectilinear map with sliding labels. *Int. J. Comput. Geom. Appl.*, 11(2):167–179, 2001.
4. D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM J. Discr. Math.*, 5(3):422–427, 1992.
5. D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.
6. E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.
7. C. K. Poon, B. Zhu, and F. Chin. A polynomial time solution for labeling a rectilinear map. *Information Processing Letters*, 65(4):201–207, 1998.
8. S.-H. Poon, C.-S. Shin, T. Strijk, T. Uno, and A. Wolff. Labeling points with weights. *Algorithmica*, 38(2):341–362, 2003.
9. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 3rd edition, 1990.
10. T. Strijk and M. van Kreveld. Labeling a rectilinear map more efficiently. *Information Processing Letters*, 69(1):25–30, 1999.