# A Simple Linear Algorithm for Computing Rectangular 3-Centers

Michael Hoffmann*
Institut für Theoretische Informatik
ETH Zürich
hoffmann@inf.ethz.ch

## Abstract

Rectangular $p$-centers of a finite planar point set $\mathcal{P}$ are the centers of at most $p$ axis-parallel congruent squares of minimal size covering $\mathcal{P}$. We give a simple linear time algorithm based on linear selection for the case $p = 3$. A linear algorithm for this problem is already known [7]. But it makes use of an $LP$-type [6] formulation of the problem with high combinatorial dimension (roughly 40) which makes it unlikely to perform well in an actual implementation. The motivation for our algorithm is such an implementation.

## 1 Theoretical Results

Let $\mathcal{P}$ be a set of points in the plane and denote its bounding box by $\mathcal{B}_\mathcal{P}$. The 3-*radius* of $\mathcal{P}$ is the minimal $\varrho \in \mathbb{R}$ such that $\mathcal{P}$ can be covered by three axis-parallel congruent squares of side length $2\varrho$. Excluding trivial cases we have $\mathcal{B}_\mathcal{P} = [x_l, x_r] \times [y_b, y_t]$, $x_l < x_r$, $y_b < y_t$. For the sake of simplicity let us furthermore assume that $\mathcal{P}$ is in general position, i.e. no two points have a common $x$– or $y$-coordinate nor the same $||\cdot||_\infty$-distance to one of the corners of $\mathcal{B}_\mathcal{P}$. We will first repeat a number of simple observations as listed in [7].

**Observation 1**

1. *We can restrict ourselves to squares that are contained in $\mathcal{B}_\mathcal{P}$.*
2. *Since each of the four line segments bounding $\mathcal{B}_\mathcal{P}$ contains at least one point from $\mathcal{P}$, we have to place a square on each of them. Consequently, since there are four segments but only three squares, one of the squares has to be placed at a corner of $\mathcal{B}_\mathcal{P}$.*

So let us assume w.l.o.g. that one of the squares is placed at the top-left corner $C_{tl}$ of $\mathcal{B}_\mathcal{P}$. If we can compute the minimal covering of this type in linear time, it can be done similarly for the other three corners and the overall minimum will just be the minimum of the four resulting coverings.

**Observation 2** *If the first square sits at $C_{tl}$, there are two possibilities for the other two squares to be placed.*

1. *Either one sits at the bottom and the other at the right side of $\mathcal{B}_\mathcal{P}$*
2. *or one is placed at the bottom-right corner $C_{br}$ of $\mathcal{B}_\mathcal{P}$ and the other at the top-left corner of the bounding box of the set of points not covered by the first square.*

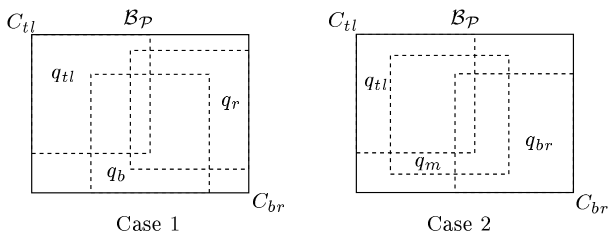


Figure 1:

Let us have a look at the latter case first, i.e. two squares $q_{tl}$ and $q_{br}$ are placed at $C_{tl}$ and $C_{br}$. Consider the process of continuously increasing the radius of both $q_{tl}$ and $q_{br}$ starting from a value of zero. At any time some of the points from $\mathcal{P}$ are covered by $S := \{q_{tl}, q_{br}\}$ and some are not. The moment we are interested in during this process is the first where those points not covered by $S$ can be covered by another square of the same radius.

*This work is partially supported by the ESPRIT IV LTR Projects No. 21957 (CGAL) and 28155 (GALIA), and by the Swiss Federal Office for Education and Science (CGAL and GALIA).

More formally define for a point $p \in \mathcal{P}$

$$\gamma(p) := \min \{ \, ||p - C_{tl}||_\infty, \; ||p - C_{br}||_\infty \, \}$$

to be the minimal radius needed to cover it by $S$ and denote by

$$\Gamma(p) := \{q \in \mathcal{P} \mid \gamma(q) > \gamma(p)\}$$

the set of points from $\mathcal{P}$ which are not covered by $S$ with radius $\gamma(p)$.
The 3-radius is now $\frac{1}{2} \min \gamma(p)$, where $p \in \mathcal{P}$ such that $\Gamma(p)$ can be covered by a square of side length $\gamma(p)$, and it can be computed easily by the following algorithm.

**Algorithm 3** (`3-Cover-1`($\mathcal{P}$))

1. *$B \leftarrow \emptyset$, $\mathcal{P}t \leftarrow \mathcal{P}$.*
2. *`while` ($|\mathcal{P}t| > 1$)*
   (a) *Compute the (lower) median $\gamma(m)$ of $\gamma(\mathcal{P}t)$.*
   (b) *Compute the quadratic bounding box $B'$ of $\Gamma(m)$.*
   (c) *`if` (sidelength$(B') > \gamma(m)$)*
   $\mathcal{P}t \leftarrow \mathcal{P}t \, \cap \, \Gamma(m)$.
   (d) *`else`*
   $\mathcal{P}t \leftarrow \mathcal{P}t \, \setminus \, \Gamma(m)$, $B \leftarrow B'$.
3. *`if` ($\gamma(p), p :\in \mathcal{P} >$ sidelength$(B \sqcup \{p\})$)*

   *Return the radius $\varrho =$ sidelength$(B \sqcup \{p\})$ and the corresponding squares $q_{tl}$, $q_{br}$ and $B \sqcup \{p\}$.*

   *`else`*

   *Return the radius $\varrho = \gamma(p)$ and the corresponding squares $q_{tl}$, $q_{br}$ and $B$.*

*where $A \sqcup B$ denotes the quadratic bounding box of $A \cup B$.*

**Lemma 4** *Algorithm `3-Cover-1`($\mathcal{P}$) computes rectangular 3-centers of $\mathcal{P}$ under the assumptions from Observation 2.2 in $\mathcal{O}(|\mathcal{P}|)$ time.*

**Proof:**
For the correctness note that the distinction in step 3 is necessary since we have been a bit too pessimistic in step 2c. The point $m$ is excluded from $\mathcal{P}t$ although it might still be the one defining the radius in the sense that it is the last one to be covered by the two corner squares. In other words the question is whether or not it is cheaper to cover $p$ with the middle square.
Considering the runtime note the following invariant that is valid after step 2a in each iteration of the loop.

$B$ is the quadratic bounding box of $\Gamma(m) \, \cap \, (\mathcal{P} \setminus \mathcal{P}t)$ .

This implies that the computation of $B'$ in step 2b can be done in $\mathcal{O}(|\mathcal{P}t|)$ time with help of $B$ (the union of two bounding boxes can be computed in constant time). It is well known that one can select from an ordered set in linear time (see e.g. [2]). Hence step 2a can be done in $\mathcal{O}(|\mathcal{P}t|)$ time as well and this is the time bound for one iteration of loop 2. Since the size of $\mathcal{P}t$ is approximately halved in each iteration, we can bound the overall runtime $T(|\mathcal{P}|)$ for Algorithm 3 by

$$T(|\mathcal{P}|) \leq c \cdot |\mathcal{P}| + T(\left\lceil \frac{|\mathcal{P}|}{2} \right\rceil) = \mathcal{O}(|\mathcal{P}|) \; .$$

□

Next we restrict our attention to the first case (see Observation 2.1) and call the three squares $q_{tl}$, $q_b$ and $q_r$, to be placed at the top-left corner, bottom side and right side of $\mathcal{B}_\mathcal{P}$ respectively. Note that it is sufficient to compute the set of points covered by $q_{tl}$ in an optimal covering, since we then can run a 2-center algorithm (see e.g. [3]) on the rest of the points to obtain a solution.
The idea of the following algorithm is to try to cover $\mathcal{P}$ with a certain radius and according to the result of this try to assign a number of points to one of the three squares. This indicates that a point is assigned to square indicates that it has to be covered by this particular square. Luckily, there is no need to keep track of all the points assigned to a square, since a set of points is covered by a square $s$ iff its bounding box is covered by $s$. Hence, what has to be stored is just one bounding box for each square. For $q_{tl}$ things are even simpler. We just have to store a lower bound $\varrho_{\min}$ for its radius, since its position is already fixed.

**Algorithm 5** (`3-Cover-2`($\mathcal{P}$))

1. *$\varrho_{\min} \leftarrow 0$, $\varrho_{\max} \leftarrow \infty$, $Q_b \leftarrow \emptyset$, $Q_r \leftarrow \emptyset$.*
2. *`while` ($|\mathcal{P}| > 6$):*
   (a) *Adjust the size $\varrho$, $\varrho_{\min} \leq \varrho \leq \varrho_{\max}$ of $q_{tl}$, such that it contains exactly $k := \left\lfloor \frac{|\mathcal{P}|}{7} \right\rfloor$ points.*
   (b) *Place $q_b$ and $q_r$ accordingly, such that no uncovered point is to the left resp. above them. Let $G := (q_b \, \Delta \, q_r) \setminus q_{tl}$ where $\Delta$ denotes the symmetric difference and $R := (q_b \, \cap \, q_r) \setminus q_{tl}$.*
   (c) *`if` (one of $\mathcal{P}$, $Q_b$ or $Q_r$ is not covered):*
      i. *$\mathcal{P} \leftarrow \mathcal{P} \setminus (q_{tl} \, \cup \, R)$.*
      ii. *$\varrho_{\min} \leftarrow \varrho$.*
   (d) *`else if` ($|G| \geq k$):*
      i. *$\mathcal{P} \leftarrow \mathcal{P} \setminus G$.*
      ii. *$Q_b \leftarrow Q_b \cup (G \cap q_b)$, $Q_r \leftarrow Q_r \cup (G \cap q_r)$.*
      iii. *$\varrho_{\max} \leftarrow \varrho$.*

(e) `else` *shrink down* $q_{tl}$ *such that – if* $q_b$, $q_r$, $G$ *and* $R$ *are set accordingly as described in the previous steps –* $R$ *as well as* $G$ *contain at least* $k$ *points. Again check for covering and continue as described above in 2c resp. 2d.*

3. *Compute the solution with radius within* $[\varrho_{\min}, \varrho_{\max}]$ *and such that* $q_b$ *covers* $Q_b$ *and* $q_r$ *covers* $Q_r$ *directly.*

First of all let us argue that the algorithm is correct in assigning the points as described.

**Theorem 6** *Algorithm 3-Cover-2(*$\mathcal{P}$*) computes rectangular* 3*-centers of* $\mathcal{P}$ *under the assumptions from Observation 2.2.*

**Proof:** The algorithm terminates, since the loop in step 2 is only entered if $|\mathcal{P}| \geq 7$, and in each iteration at least $\left\lfloor \frac{|\mathcal{P}|}{7} \right\rfloor \geq 1$ points are discarded.
Now consider the situation as depicted in figure 2, where the sets $G$ and $R$ are drawn light resp. dark shaded. If $\mathcal{P}$ is not covered, the size of the squares has to be increased in order to produce a covering. This implies that the set of points covered by $q_{tl}$ will be a superset of the points covered presently. Consequently the squares $q_b$ and $q_r$ will not move left– resp. top wards. Since there is no point in letting them pass the right resp. bottom border of $\mathcal{B}_\mathcal{P}$, the points presently in $R$ will remain covered by both $q_b$ and $q_r$.

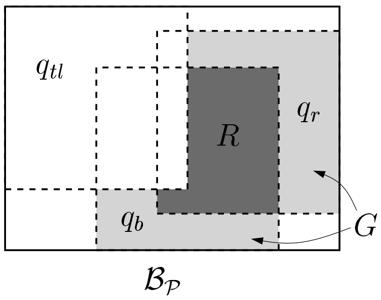


Figure 2:

On the other hand, if $\mathcal{P}$ is covered, the radius possibly has to be decreased to find a covering with smaller side length. If $q_{tl}$ shrinks down, some points might get exposed that have been covered by $q_{tl}$ before. This causes $q_b$ and $q_r$ to move left– resp. top wards in order to cover them. But $q_b$ and $q_r$ will never move to the right resp. below the current position anymore. Hence, in order to produce a covering, the set of points covered by $q_b$ resp. $q_r$ has to be a superset of the set covered now. □

The crucial point here is to show that each of these steps in Algorithm 5 can be handled in $\mathcal{O}(|\mathcal{P}|)$ time. Then the overall linearity easily follows by observing that we discard $\left\lfloor \frac{1}{7} \right\rfloor$ of the points in each iteration. Thus the runtime $T(n)$ of the algorithm for input size $n := |\mathcal{P}|$ can be expressed as

$$T(n) \leq c \cdot n + T\left(n - \left\lfloor \frac{n}{7} \right\rfloor\right) = \mathcal{O}(n) \ .$$

**Lemma 7** *One step of Algorithm 5 can be handled in* $\mathcal{O}(n)$ *time where* $n := |\mathcal{P}|$*.*

**Proof:** If we order $\mathcal{P}$ according to $||\cdot - C_{tl}||_\infty$, step 2a is just again an instance of the selection problem and can thus be computed in linear time. Obviously, the same time bound suffices for step 2b and the covering test as well as for the computation of $R$ and $G$ whereas the other operations in step 2c and 2d are constant time. We will show below that step 2e requires linear time as well. □

The goal in step 6 is to shrink down $q_{tl}$ such that $R$ as well as $G$ contain at least $k = \left\lfloor \frac{|\mathcal{P}|}{7} \right\rfloor$ points. Let us have a closer look at the set $R$. The configuration corresponds to a covering of $\mathcal{P}$ and neither $q_{tl}$ nor $G$ contain more than $k$ points (for $q_{tl}$ remember the general position assumption). Hence $R$ contains at least $\left\lceil \frac{5 \cdot n}{7} \right\rceil$ points.
Consider the vertical line $l$ such that $\left\lfloor \frac{n}{7} \right\rfloor$ points lie in the intersection of the closed halfplane to the left of $l$ with $R$ and similarly the lines that cut off $\left\lfloor \frac{n}{7} \right\rfloor$ points from the right, bottom and top of $R$. Each of these lines can be computed in $\mathcal{O}(n)$ time using a standard selection algorithm with the appropriate ordering on $x$– resp. $y$-coordinates. Note that the set $B$ that remains in the middle of $R$ still contains at least $\left\lceil \frac{n}{7} \right\rceil$ points.

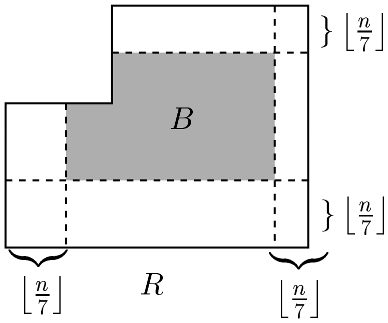


Figure 3:

The idea is to shrink the squares continuously until at some point at least one of $q_b$ and $q_r$ does not cover the region $B$ anymore. Of course, we do not simulate this continuous process, but replace it by a discrete process such that at any step there is a new point from $\mathcal{P}$ on the right or bottom side of $q_{tl}$. Recall that due to the general position assumption there will never be two points from $\mathcal{P}$ on these two sides at the same moment. Let us

call these moments, where $q_{tl}$ hits a point from $\mathcal{P}$ during the shrink-process, *events* and associate an event with the point that causes it.
Once we have reached the event described above (the first event where at least one of $q_b$ and $q_r$ does not cover $B$ anymore) which we will call $E$ now, we check for covering. If $\mathcal{P}$ is still covered, we can discard the points from $G$. These are at least $\lfloor \frac{n}{7} \rfloor$ by definition of $B$. On the other hand, if $\mathcal{P}$ is not covered anymore, we consider the event $P$ immediately before $E$. That is, where the point $P \in \{q \in \mathcal{P} \mid ||q - C_{tl}||_\infty > ||E - C_{tl}||_\infty\}$ that has minimal distance to $C_{tl}$ determines $q_{tl}$. If $\mathcal{P}$ is still not covered at $P$, we can discard the points from $R$, which are at least $\lfloor \frac{n}{7} \rfloor$ by definition of $B$. Otherwise we precisely know the set of points to be covered by $q_{tl}$ in an optimal covering, except for $P$. The question whether or not to include $P$ can be solved by calling the 2-center algorithm on $(\mathcal{P} \setminus q_{tl}) \cup \{p\}$. If the resulting 2-covering is smaller than $||P - C_{tl}||_\infty$, there is not need to cover $P$ with $q_{tl}$ and otherwise we have to do so.
The remaining question is how to find this moment $E$. This amounts to a search on the set $\mathcal{P} \cap q_{tl}$ ordered by $\delta := || \cdot - C_{tl}||_\infty$. While we do not know this ordering explicitly and cannot afford to compute it, we can again apply the standard technique for linear selection. This is formulated in the following algorithm which has four parameters, the set $S$ to be searched (initially $\mathcal{P} \cap q_{tl}$), the set $B$ that should not be enclosed by both $q_r$ and $q_b$, and finally $\mathrm{pos}(q_b)$ and $\mathrm{pos}(q_r)$ denoting the current position of $q_b$ resp. $q_r$ during the process.

**Algorithm 8** ($\texttt{search\_E}(S, B, \texttt{pos}(q_b), \texttt{pos}(q_r))$)

*while* $(|S| > 1)$

1. *Compute the (upper) median $m$ of $S$ w.r.t. $\delta$ using a standard linear selection algorithm.*
2. *Compute the positions $pos^m(q_b)$ of $q_b$ and $pos^m(q_r)$ of $q_r$ at event $m$.*
3. *If $q_b$ and $q_r$ enclose $B$ at $m$, recurse with $\texttt{search\_E}(S_{<m}, B, pos^m(q_b), pos^m(q_r))$ where $S_{<m} := \{s \in S \mid s < m\}$.*
4. *Otherwise recurse with $\texttt{search\_E}(S_{\geq m}, B, pos(q_b), pos(q_r))$ where $S_{\geq m} := \{s \in S \mid s \geq m\}$.*

We will now show the linear runtime bound for this algorithm thereby completing the proof of Lemma 7.

**Lemma 9** *Algorithm 8 computes the event $E$ in $\mathcal{O}(|S|)$ time.*

**Proof:** Step 1 together with the computation of $S_{\geq m}$ resp. $S_{<m}$ needs $\mathcal{O}(|S|)$ time while step 3 can be handled in constant time. For step 2 note that the position of $q_r$ at $m$ is either determined by some point in $S_{\geq m}$ or stays the same as before ($\mathrm{pos}(q_r)$). Similarly for $q_b$. Hence both can be determined in $\mathcal{O}(|S|)$ time as well. Since $|S|$ is halved in each step, the running time $T(|S|)$ can be expressed by the recursion

$$T(n) \leq c \cdot |S| + T\left(\left\lceil \frac{|S|}{2} \right\rceil\right) = \mathcal{O}(|S|) \ .$$

□

**Theorem 10** *Rectangular 3-centers of a finite point set $\mathcal{P}$ can be computed in $\mathcal{O}(|\mathcal{P}|)$ time using Algorithm 3 and 5.*

**Proof**: Lemma 4, Theorem 6 and Lemma 7 □

## 2 Implementation

There are a few things to note regarding an actual implementation of our algorithm. First one has to get rid of the general position assumptions. This is not really problematic here and can be solved by imposing some total ordering on the points e.g. with same $|| \cdot ||_\infty$-distance to a corner (*perturbation*). It turns out that this ordering does not even have to be computed explicitly, but this is a minor detail. Second one has to implement the algorithm for linear selection. Although this can be done in deterministic linear time as noted, the constants in this linear term are rather large. Thus from a practical point of view the standard randomized selection algorithm that needs expected linear time seems preferable. This is what we used in our implementation.

### 2.1 Test Data

In order to test the performance of an implementation, one has to have a set of test data. An important question in this context is always, for what kind of input data one wants to evaluate the algorithm and sometimes also how to generate this data efficiently.
In our case the input consists of a set of points, so one of the most simple test data would be a set drawn uniformly at random from the unit square. But as the number of points increases, the 3-coverings of these random point sets tend to consist of three almost coinciding squares with side length close to one. So in some sense these are very special input sets where we do not expect to find a "nice" way to divide the points into three clusters.
Therefore we have experimented with a second type of random point sets. These consist of points drawn uniformly from three congruent squares of side length $\frac{1}{4}$. The squares in turn are placed uniformly at random inside the unit square. In these cases we always expect a "nice" 3-clustering, so this is where doing a 3-covering really makes sense.

## 2.2 Heuristics

We have also added two simple heuristics to speed up the algorithm in many cases.
In step 2a of Algorithm 5 the size of the corner square is adjusted such that it contains $\frac{1}{7}$ of the points. This leads to a rather slow convergence if the square contains many points in the optimal covering.

**Heuristic 11 (Prefilter)** *To speed up these cases, we first try to cover half of the points with the corner square. If covering is not possible, these points are discarded as in step 2c and we continue with cutoff $\frac{1}{2}$. But as soon as we get a covering, we switch to $\frac{1}{7}$ for the rest of the algorithm.*

In the description above we restricted ourselves to the case that one square is placed at the top-left corner. Hence we have to run Algorithm 3 two– and Algorithm 5 four times to find the overall best covering. As soon as the first of these runs has ended, the resulting radius serves as an upper bound for the final result. In the following it does not make sense to compute coverings with a radius exceeding that bound.

**Heuristic 12 (Check)** *Thus we check at the start of each run whether we can beat the current bound in the actual setting and proceed only if this is possible.*

Obviously neither of these heuristics changes the asymptotic behaviour. While in some cases either one of them might result in a loss of performance, we expect this slowdown to be small compared to the overall runtime. Regarding the test data as described in the previous section, we expect

- prefiltering to improve the performance significantly for point sets drawn uniformly from the unit square,
- prefiltering to have less positive influence for point sets drawn uniformly from three clusters as the side length of the cluster squares decreases *and*
- checking to improve the performance in most cases independently from the input data.

For the check-heuristic note that there is a gain whenever we find good solutions first. Hence, if we randomize the order in which the different cases are handled, we can expect a gain. We only loose in the rare case that we encounter the solutions in worst-to-best order.
Indeed, our expectations can be confirmed by looking at the experimental results depicted in Figure 4 and 5. The $x$-axis describes the input size in units of 1000 points and the $y$-axis gives the corresponding runtime in seconds[1] for the algorithm with and without the two heuristics.

[1]on an SGI Indigo2 with 175MHz MIPS R10000(IP28) processor

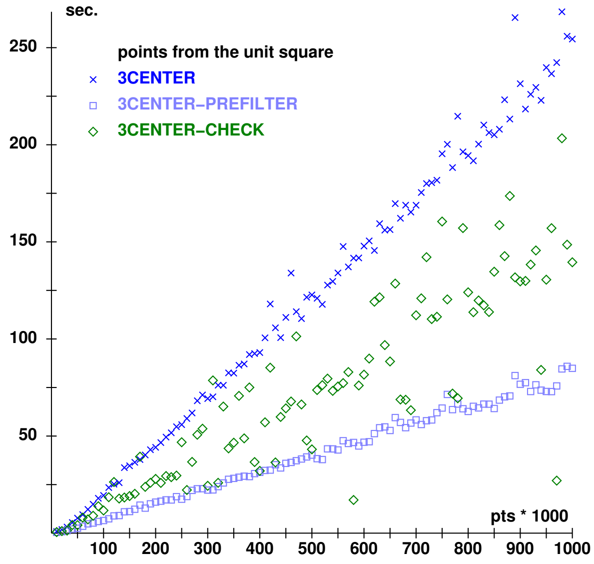


Figure 4: Comparison of heuristics with input points from the unit square.

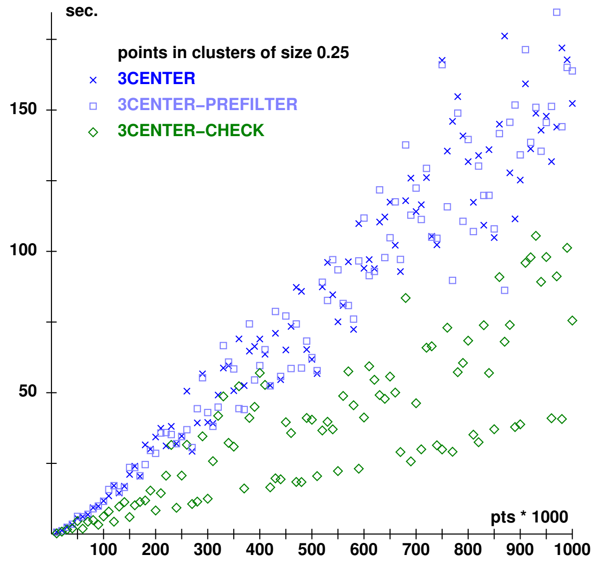


Figure 5: Comparison of heuristics with input points from three clusters.

It turns out that for the clustered input data the effect of prefiltering is about half and half positive or negative, but even for very small cluster sizes (results not shown here) the slowdown is small compared to the overall runtime, such that the general use of both heuristics is justified.

### 2.3 Comparison to the algorithm in CGAL

The CGAL[1] library contains an $\mathcal{O}(n \cdot \log n)$ implementation for computing rectangular 2–4-centers based on searching in sorted matrices [4],[5]. We have compared the performance of our new algorithm compared to the 3-center algorithm currently in CGAL expecting to beat it at least for sufficiently large point sets.
It turned out that the linear algorithm outperforms the matrix-search even for small point sets from our test data, at least as soon as we apply both heuristics described in the previous section. Without prefiltering the matrix-search is faster on point sets drawn uniformly from the unit square. But as already noted above, these point sets that cannot be three-clustered nicely are somewhat special. It seems that the matrix-search is fairly efficient if the value one searches for is close to the maximum value of the set.
The results of the test runs are shown below in Figure 6 and 7. Again the $x$-axis describes the input size in units of 1000 points and the $y$-axis gives the corresponding runtime in seconds for the different algorithms, i.e. the $p$-center algorithm from CGAL and our new 3-center algorithms with both heuristics as described above.

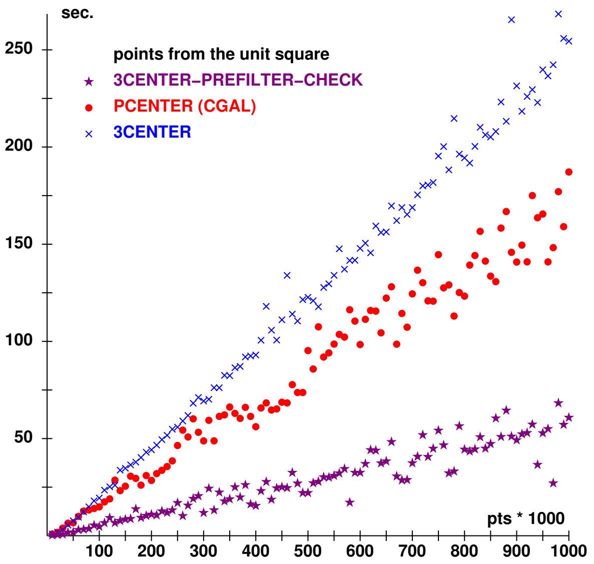


Figure 6: Comparison to $p$-center algorithm with input points from the unit square.

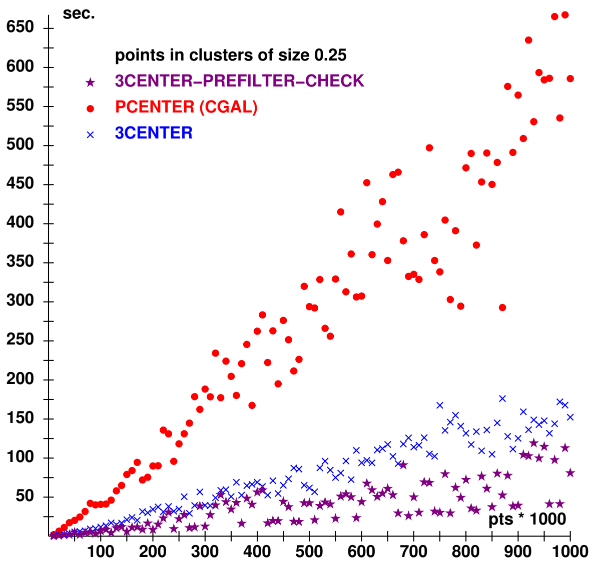


Figure 7: Comparison to $p$-center algorithm with input points from three clusters.

## 3 Conclusion

We have described and implemented a new linear algorithm for the rectangular 3-center problem and two heuristics to improve its performance in practice. A number of tests on certain randomly generated test sets have been made to compare the new algorithm to the existing $\mathcal{O}(n \cdot \log n)$ implementation in CGAL. In all tests the linear algorithm outperformed the $\mathcal{O}(n \cdot \log n)$ algorithm with factors ranging from 3 to 8. Thus it is very likely that the new algorithm will appear in the next release of the CGAL library.

## References

[1] CGAL. Computational geometry algorithms library. `http://www.cs.uu.nl/CGAL/`.

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

[3] Drezner, Z. On the rectangular $p$-center problem. *Naval Res. Logist. Q. 34* (1987), 229–234.

[4] Frederickson, G. N., and Johnson, D. B. Finding $k$th paths and $p$-centers by generating and searching good data structures. *J. Algorithms 4* (1983), 61–80.

[5] Frederickson, G. N., and Johnson, D. B. Generalized selection and ranking: sorted matrices. *SIAM J. Comput. 13* (1984), 14–30.

[6] Matoušek, J., Sharir, M., and Welzl, E. A subexponential bound for linear programming. *Algorithmica 16* (1996), 498–516.

[7] Sharir, M., and Welzl, E. Rectilinear and polygonal $p$-piercing and $p$-center problems. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.* (1996), pp. 122–132.