Sequential Dependency Computation via Geometric Data Structures

Gruia Calinescu * Howard Karloff†

Abstract

We are given integers $0 \leq G_1 \leq G_2 \neq 0$ and a sequence $S_N = \langle a_1, a_2, ..., a_N \rangle$ of N integers. The goal is to compute the minimum number of insertions and deletions necessary to transform S_N into a valid sequence, where a sequence is *valid* if it is nonempty, all elements are integers, and all the differences between consecutive elements are between G_1 and G_2. For this problem from the database theory literature, previous dynamic programming algorithms have running times $O(N^2)$ and $O(A \cdot N \log N)$, for a parameter A unrelated to N. We use a geometric data structure to obtain a $O(N \log N \log \log N)$ running time.

1 Introduction

Golab, Karloff, Korn, Saha, and Srivastava introduce the following problem in VLDB 2009 [3]: We are given integers $0 \leq G_1 \leq G_2 \neq 0$ and a (not necessarily sorted) sequence $S_N = \langle a_1, a_2, ..., a_N \rangle$ of N integers. The goal is to compute the minimum number of insertions and deletions necessary to transform S_N into a valid sequence, where a sequence is *valid* if it is nonempty, all elements are integers, and all the differences between consecutive elements are between G_1 and G_2. That is, $\langle b_1, b_2, ..., b_M \rangle$ is valid if $M \geq 1$ and for all $i \in \{1, \dots, M-1\}$, $G_1 \leq b_{i+1} - b_i \leq G_2$. We term the problem Gap Dependency.

An example instance of Gap Dependency and its solution has $G_1 = 4$, $G_2 = 6$, and $\langle 1, 7, 5, 9, 12, 25, 31, 30, 34, 40 \rangle$ as the (invalid) input sequence. A feasible solution deletes the first five elements and the seventh element, resulting in the valid sequence $\langle 25, 30, 34, 40 \rangle$, at cost 6. A better feasible solution, of cost 5, starts by deleting 12 and inserting 15 and 20 in its place, obtaining the sequence $\langle 1, 7, 5, 9, 15, 20, 25, 31, 30, 34, 40 \rangle$, which is still not valid since $5 - 7 < 4$ and $30 - 31 < 4$. After deleting 7 and 31, we obtain the valid sequence $\langle 1, 5, 9, 15, 20, 25, 30, 34, 40 \rangle$. Yet another solution of cost 5 deletes $5, 9, 31$ (resulting in sequence $\langle 1, 7, 12, 25, 30, 34, 40 \rangle$, which is invalid since $25 - 12 > 6$), followed by inserting 16 and 20 between 12 and 25.

Golab et al. [3] present an algorithm with running time $O(\frac{G_2}{G_2 - G_1} N \log N)$ for $G_2 > G_1 > 0$ (and $O(N \log N)$ if $G_1 = 0$ or $G_1 = G_2$). This is pseudopolynomial running time. Implicit in [3] is also a $O(N^2)$-time algorithm. In this paper we give a $O(N \log N \log \log N)$-time algorithm for $G_2 > G_1 > 0$, by exploiting a surprising connection to geometric data structures.

2 Preliminaries

We include definitions and results from [3]. Given a sequence $S_N = \langle a_1, a_2, ..., a_N \rangle$, define S_i to be the prefix $S_i = \langle a_1, a_2, ..., a_i \rangle$, and $OPT(i)$ to be the value of the Gap Dependency optimum with input S_i.

Given a sequence $\langle a_1, a_2, ..., a_N \rangle$ of integers, for $i = 1, 2, ..., N$, let $v = a_i$ and define $T(i)$ to be the minimum number of insertions and deletions one must make to $\langle a_1, a_2, ..., a_i \rangle$ in order to convert it into a valid sequence ending in the number v.

Computing $OPT(N)$ from the $T(i)$'s can be done as follows. $OPT(N) = \min_{0 \leq r \leq N-1} \{r + T(N - r)\}$, as proven in Claim 1.

Claim 1 *[Claim 3 of [3]] The minimum number $OPT(i)$ of insertions and deletions required to convert sequence S_i into a valid one is given by $\min_{0 \leq r \leq i-1} \{r + T(i-r)\}$. Furthermore, $OPT(i)$ can be calculated inductively by $OPT(1) = 0$ and $OPT(i) = \min\{1 + OPT(i-1), T(i)\}$ for all $i \geq 2$.*

In order to show how to compute the $T(i)$'s, we need the following definition from [3]:

Definition 1 Define $dcost(d)$, for $d = 0, 1, 2, ...$, to be the minimum number of integers one must append to the length-1 sequence $\langle 0 \rangle$ to get a valid sequence ending in d, and ∞ if no such sequence exists.

For example, if $G_1 = 4$ and $G_2 = 6$, then $dcost(7) = \infty$. Furthermore, $dcost(8) = 2$, uniquely obtained by appending 4 and 8. We compute $dcost$ very differently. Precisely, we use existing geometric data structures. Instead of this lemma:

Lemma 1 (Lemma 6 of [3]) *If $G_1 = 0$, then $dcost(d) = \lceil d/G_2 \rceil$. Otherwise, $dcost(d) = \lceil d/G_2 \rceil$ if $\lceil (d+1)/G_1 \rceil > \lceil d/G_2 \rceil$ and ∞ otherwise,*

*Department of Computer Science, Illinois Institute of Technology; calinescu@iit.edu. Research supported in part by NSF grant CCF-0515088.

†AT&T Labs–Research, 180 Park Ave., Room C231, Florham Park, NJ 07932, USA; howard@research.att.com.

we use the method of the following section. We do so since the previous dynamic programs [3] may use the lemma for $\Omega(\min\{N^2, \frac{G_2}{G_2-G_1} N \log N\})$ values of d, even though *dcost* can be computed in constant time.

The $O(N^2)$ algorithm of [3] follows in a rather straightforward way from Claim 1, the lemma above, and Theorem 2 which appears later. We refer to [3] for the more sophisticated $O(\frac{G_2}{G_2-G_1} N \log N)$ algorithm.

3 The new algorithm for computing the $T(i)$-values

In this paper we will assume that $0 < G_1 < G_2$.

What differentiates this paper from [3] is the use of a fast geometric data structure to calculate the $T(i)$'s, in amortized time $O(\log N \log\log N)$ each. We show how the recurrence used in [3] can be modified to make use of a data structure allowing fast 2-dimensional range minimum queries, and thereby to decrease the running time from $O(\min\{N^2, \frac{G_2}{G_2-G_1} \cdot N \log N\})$ to $O(N \log N \log\log N)$. (This is only an improvement, of course, if $\frac{G_2}{G_2-G_1} > \log\log N$.)

We assume all the values a_i are nonnegative. (Otherwise, let $m = \min_i a_i$ and set $a_i := a_i - m$.) For each j, create point $P_j = (x_j, y_j)$ with $x_j = a_j \bmod G_2$ and $y_j = \lfloor a_j / G_2 \rfloor$. Two values of j can have points P_j with the same coordinates; we treat the points P_j as distinct. Let $\Delta := G_2 - G_1 > 0$.

For given i, define two regions in the two dimensional Euclidean plane as follows (see Figure 1 for an example). Let $q_i(x)$ be the linear map

$$q_i(x) = y_i - (x - (x_i - G_1))/\Delta$$

and let Q_i be the halfspace

$$Q_i = \{(x, y) \ : \ y \leq q_i(x)\}.$$

Let $r_i(x)$ be the linear map

$$r_i(x) = y_i - (x - x_i)/\Delta$$

and let R_i be the intersection of the halfspaces

$$\{(x, y) | y \leq r_i(x)\}$$

and

$$\{(x, y) | x \geq x_i\},$$

and last, let $R_i^* = R_i \setminus \{(x_i, y_i)\}$.

(It will be crucial later that all the lines $r_i(x)$, over all i, and all lines $q_i(x)$, over all i, have the same slope. These facts will allow us to find *one* affine transformation converting, for all i, Q_i into a halfspace with *axis-parallel* bounding line, and R_i into an intersection of two halfspaces, whose bounding lines are orthogonal *axis-parallel* lines.)

Our algorithm relies on the following theorem from [3].

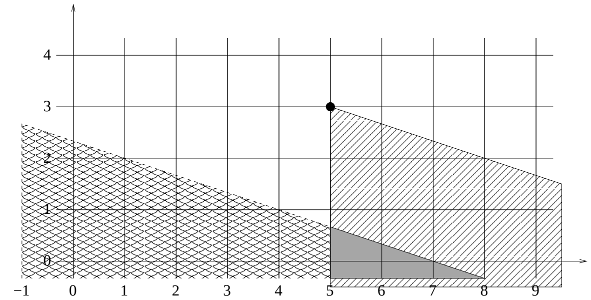

Figure 1: Here $G_2 = 10$, $G_1 = 7$, $a_i = 35$. The point P_i is given by the small dark circle. R_i and Q_i are unbounded and we only show their relevant parts—where other points P_j could be located. R_i is the region on the right, colored using a diagonal pattern. Q_i is the region on the left, colored using a doubly diagonal pattern. Where the regions intersect, we use a solid pattern.

Theorem 2 *[3] Fix $i \geq 2$. Assume $G_1 > 0$. Define $m := \min_{j<i, a_j<a_i}\{T(j)+(i-1-j)+[dcost(a_i-a_j)-1]\}$. Then $T(i) = \min\{i-1, m\}$.*

For intuition only, we explain the recurrence. To end an optimal subsequence with a_i, we either delete the first $i-1$ elements, or, with j being such that $j < i$ and $a_j < a_i$, take the optimal subsequence ending with a_j, delete the $i-1-j$ elements between a_j and a_i, and insert $dcost(a_i - a_j) - 1$ elements between a_j and a_i. (The "-1" is here since, as defined, $dcost(d)$ also inserts "d", while we do not have to insert "a_i".)

We will prove the following theorem by relating it to Theorem 2.

Theorem 3 *Fix $i \geq 2$. Define*

$$r := \min_{j \,:\, j<i, P_j \in R_i^*} \{T(j) + (i-j-1) + (y_i - y_j) - 1\}$$

and

$$q := \min_{j \,:\, j<i, P_j \in Q_i} \{T(j) + (i-j-1) + (y_i - y_j)\}.$$

Then $m = \min\{q, r\}$.

To prove Theorem 3, we need Claim 2. Recall that the x-coordinate of each P_k is at most $G_2 - 1$.

Claim 2 *1. For any $k < i$, [$a_k < a_i$ and $dcost(a_i - a_k) < \infty$] if and only if $P_k \in Q_i \cup R_i^*$.*

2. If $P_k \in R_i^*$, then $a_k < a_i$ and $dcost(a_i - a_k) = y_i - y_k$.

3. If $P_k \in Q_i \setminus R_i$, then $a_k < a_i$ and $dcost(a_i - a_k) = y_i - y_k + 1$.

We will prove Claim 2 in a moment.

Proof of Theorem 3. We need to prove that

$$\min\{q,r\} = \min_{j<i, a_j<a_i} \{T(j) + (i-1-j)$$
$$+[dcost(a_i - a_j) - 1]\}.$$

By part 1 of Claim 2, the two minima are infinite on exactly the same set. Using this and the fact that $P_i \notin Q_i$ (because $q_i(x_i) = y_i - G_1/\Delta$ and $G_1 > 0$ so that $y_i > q_i(x_i)$),

$$m = \min_{j<i, P_j \in Q_i \cup R_i^*} [T(j) + (i-1-j) + dcost(a_i - a_j) - 1]$$
$$= \min\{ \min_{j<i, P_j \in R_i^*} [T(j) + (i-1-j) + dcost(a_i - a_j) - 1],$$
$$\min_{j<i, P_j \in Q_i \setminus R_i} [T(j) + (i-1-j) + dcost(a_i - a_j) - 1],$$
$$\min_{j<i, P_j \in Q_i \cap R_i} [T(j) + (i-1-j) + dcost(a_i - a_j) - 1]\}.$$

Now we use parts 2 and 3 of Claim 2 and the fact that $P_i \notin Q_i$ to infer that m equals

$$\min\{ \min_{j<i, P_j \in R_i^*} [T(j) + (i-1-j) + y_i - y_j - 1],$$
$$\min_{j<i, P_j \in Q_i \setminus R_i} [T(j) + (i-1-j) + y_i - y_j],$$
$$\min_{j<i, P_j \in Q_i \cap R_i} [T(j) + (i-1-j) + y_i - y_j - 1]\}.$$

Letting

$$A := \min_{j<i, P_j \in R_i^*} [T(j) + (i-1-j) + y_i - y_j - 1],$$

$$B := \min_{j<i, P_j \in Q_i \setminus R_i} [T(j) + (i-1-j) + y_i - y_j],$$

and

$$C := \min_{j<i, P_j \in Q_i \cap R_i} [T(j) + (i-1-j) + y_i - y_j - 1],$$

we want to show that $\min\{A, B, C\} = \min\{q, r\}$. Since $r = A$ and $q = \min\{B, C+1\}$, $\min\{q, r\} = \min\{A, \min\{B, C+1\}\} = \min\{A, B, C+1\}$. We want to show that $\min\{A, B, C\} = \min\{A, B, C+1\}$, which follows from the fact that $A \le C$. □

Sketch of proof of Claim 2. Note that $a_i = x_i + y_i G_2$ and $a_k = x_k + y_k G_2$.

Let $I_k = [kG_1, kG_2]$ for $k \ge 0$. It is easy to see that $I_k \cap \mathbb{Z}$ is precisely the set of all integers which can be written as the sum of exactly k integers all between G_1 and G_2. Then $dcost(d)$ is the minimum k such that $d \in I_k$, if one exists, and ∞ otherwise. In other words, here is a way to compute $dcost(d)$ for all d, in principle:

Algorithm **Simpledcost**:

- Set $dcost(d) = \infty$ for all $d \ge 0$.
- For $k = 0, 1, 2, ...,$ do:
  - Set $dcost(d) = k$ for all $d \in I_k$, unless $dcost(d)$ was already defined.

We will show that the three statements in the claim are obtained in effect by "running" algorithm **Simpledcost** above.

Label the lattice points $0, 1, 2, ..., a_i$, starting by labeling the point $P_i = (x_i, y_i)$ "0", and then moving leftward, labeling points with successive integers, until a point $(0, y)$ on the y-axis is reached, and (after labeling that point) continuing with point $(G_2 - 1, y - 1)$. The point labeled "a_i" will be the origin $(0, 0)$, since the top row has $x_i + 1$ labeled points, and each of the other y_i rows has G_2 points, or $1 + a_i$ points in total, as desired.

For all $y \in \{0, 1, 2, ..., y_i\}$, the point (x_i, y) is labeled $(y_i - y)G_2$, which is the right endpoint of interval $I_{y_i - y}$.

Now execute the following:

For $l = 0, 1, 2, ..., y_i + 1$, do:

- Starting at point $(x_i, y_i - l)$ and continuing for $|I_l| = l \cdot (G_2 - G_1)$ additional steps, move rightward by one lattice point each time;

  however, if a point $(G_2 - 1, y)$ is hit, then after visiting that point, visit the point $(0, y+1)$ next and afterward continue proceeding rightward as before. (Every visited point (x, y) has $y \ge -1$.)
- Assign *dcost* equal to l for each point visited, unless its *dcost* was already assigned or its second coordinate was negative.

The points with nonnegative second coordinate visited during iteration l are exactly those whose labels are in I_l, so we are in effect executing algorithm **Simpledcost**. In other words, the existence of a point with nonnegative second coordinate with label l and assigned *dcost* d means that $dcost(l) = d$, and the existence of such a point with label l and no *dcost* means that $dcost(l) = \infty$.

(As an example, look at Figure 1. $I_0 = [0]$ and only $(5, 3)$ is assigned *dcost* 0. $I_1 = [7, 10]$ and the lattice points with *dcost* 1 are $(5, 2), (6, 2), (7, 2), (8, 2)$. $I_2 = [14, 20]$ and the lattice points with $dcost = 2$ are $(5, 1)$, $(6, 1)$, $(7, 1)$, $(8, 1)$, $(9, 1)$, $(0, 2)$, $(1, 2)$. $I_3 = [21, 30]$ and the lattice points with $dcost = 3$ are $(5, 0), (6, 0), (7, 0), (8, 0), (9, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1)$. $I_4 = [28, 40]$ and the lattice points with *dcost* 4 are $(0, 0), (1, 0), (2, 0), (3, 0), (4, 0)$.)

The following crucial statements are easy to verify. All the points assigned a finite *dcost* are in $Q_i \cup R_i$, and all such points P_k in the nonnegative quadrant get a finite *dcost*. If $P_k \in R_i^*$, then $a_k < a_i$, since $r_i(x)$ has negative slope. If $P_k \in Q_i \setminus R_i$, then, since $q_i(0) = y_i + (x_i - G_1)/\Delta \le y_i + [(G_2 - 1) - G_1]/\Delta = y_i + (\Delta - 1)/\Delta < y_i + 1$, all $P_j \in Q_i$ have $y_j \le y_i$ and hence $a_j < a_i$.

Because we assign *dcost* equal to l for points in row $y_i - l$ in R_i, as well as some to the left in Q_i in row $y_i - l + 1$, we infer that $dcost(a_i - a_k) = y_i - y_k$ if $P_k \in R_i$, and that $dcost(a_i - a_k) = y_i - y_k + 1$ if $P_k \in Q_i \setminus R_i$. □

Here is our geometric algorithm to compute the $T(i)$'s. Recall that before defining Q_i and R_i^*, for each j, we defined points $P_j = (x_j, y_j)$ with $x_j = a_j \bmod G_2$ and $y_j = \lfloor a_j / G_2 \rfloor$.

- $T(1) := 0$ and $z_1 := T(1) - 1 - y_1$.
- For $i := 2, 3, ..., n$, do
  - $r := i + y_i - 2 + \min_{j<i\,:\,P_j \in R_i^*} z_j$.
  - $q := i + y_i - 1 + \min_{j<i\,:\,P_j \in Q_i} z_j$.
  - $T(i) := \min\{i - 1, r, q\}$.
  - $z_i := T(i) - i - y_i$.

The running time of this algorithm is $O(n)$ plus the time to do the $2n$ mins involved in the definitions of m_2 and m_3. The idea is to use a geometric data structure to do each min in time $O(\log N \log\log N)$, for $O(N \log N \log\log N)$ time overall. In order to use a standard geometric data structure, we will have to convert each of the regions Q_i (a halfspace) and R_i (an intersection of two halfspaces) into a halfspace with axis-parallel boundaries, and into an orthant (an intersection of two halfspaces with axis-parallel boundaries), respectively.

The algorithm requires one to find $\min_{j<i\,:\,P_j \in R_i^*} z_j$ and $\min_{j<i\,:\,P_j \in Q_i} z_j$. It is an annoyance that the algorithm needs a minimum over $P_j \in R_i^*$ rather than over $P_j \in R_i$. Were the desired minimum over $P_j \in R_i$, one would just apply to all points the affine transformation T mapping $(x, y) \rightarrow (x, y + x/\Delta)$. This affine transformation maps points $(x, q_i(x)) = (x, (y_i + (x_i - G_1)/\Delta) - x/\Delta)$ on the bounding line of Q_i to points $(x, (y_i + (x_i - G_1)/\Delta))$, which are on a horizontal line. The same affine transformation maps points $(x, r_i(x)) = (x, (y_i + x_i/\Delta) - x/\Delta)$ on the "diagonal" bounding line of R_i to $(x, y_i + x_i/\Delta)$, another horizontal line, and maps points (x_i, y) on R_i's vertical bounding line to $(x_i, y + x_i/\Delta)$, the same vertical line. This means that the question, "Is $(x, y) \in Q_i$?" could be answered, in the transformed space, by asking if $T(x, y)$ is on or below a horizontal line, and "Is $(x, y) \in R_i$?" could be answered in the transformed space by asking if $T(x, y)$ is on or to the right of a vertical line and on or below a horizontal one.

Unfortunately, though, the min is over $P_j \in R_i^*$ instead of over R_i. We now exploit the fact that all the (untransformed) query points are of the form $(x, y) \in \mathbb{N}^2$, $x \leq G_2 - 1$. It suffices to make an affine transformation which correctly answers queries about these points.

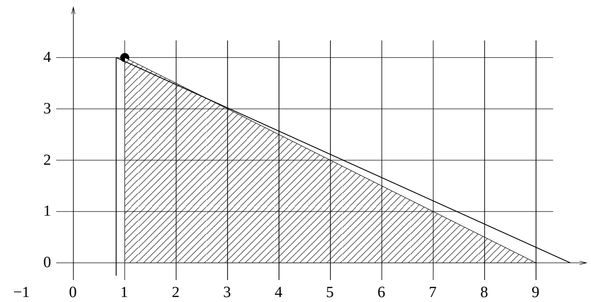

Figure 2: Here P_i is the solid point, $\Delta = 2$, the relevant part of R_i is given by the shaded area, and R_i''s bounding lines are thicker.

The idea is to replace each line $q_i(x)$ by a line $q_i'(x)$ which very closely tracks $q_i(x)$ (and to define $Q_i' = \{(x, y) | y \leq q_i'(x)\}$) and (see Figure 2 for intuition) to replace the line $r_i(x)$ by a line $r_i'(x)$ which very closely tracks $r_i(x)$, and to replace the line $x = x_i$ by $x = x_i - \epsilon$ (and to define $R_i' = \{(x, y) | (x \geq x_i - \epsilon) \wedge (y \leq r_i'(x))\}$) (for a small $\epsilon > 0$) such that (1) all lines $q_i'(x)$ over all i and $r_i'(x)$ over all i have the same slope, and (2) a point $P \in \mathbb{N}^2$ with first coordinate at most $G_2 - 1$ is in Q_i if and only if $P \in Q_i'$, and (3) a lattice point P with first coordinate at most $G_2 - 1$ is in R_i^* if and only if $P \in R_i'$.

This is done as follows. Let $h = \lceil G_2/\Delta \rceil$. The line $y = r_i(x)$, which we will call L_0, passes through $P_i = (x_i, y_i)$ and $Z := (x_i + h\Delta, y_i - h)$, since it has slope $-1/\Delta$. Consider the line segment corresponding to x-coordinates in interval $I = [x_i, x_i + h\Delta]$. (Clearly $x_i + h\Delta \geq G_2$.) For any $x \in I$, the lowest lattice point (x, y) *strictly* above the line segment is at least $1/\Delta$ above it. This means that if we hold P_i fixed and raise the right endpoint by $1/(2\Delta)$—in other words, consider the line L_1 passing through P_i and $Z' = (x_i + h\Delta, y_i - h + 1/(2\Delta))$—then "raising" the line segment causes it to "pass through" no lattice points. (The slope $\gamma := (-h + 1/(2\Delta))/(h\Delta) = -1/\Delta + 1/(2h\Delta^2)$ of L_1 does not depend on i.) Clearly, between $x = x_i$ and $x = x_i + h\Delta$, L_1 passes through no lattice points except P_i, and furthermore, the minimum distance upward from any point on L_1, whose x-coordinate is integral, to a lattice point is at least $1/\Delta - 1/(2\Delta) = 1/(2\Delta)$. In addition, the minimum distance *downward* from any point on L_1 in that interval to a lattice point other than P_i is at least $(1/(2\Delta))/(h\Delta) = 1/(2h\Delta^2)$, since the interval has length $h\Delta$.

Now simply "lower" L_1 uniformly by $\tau := 1/(4h\Delta^2)$ to get a new line L_2 which is below P_i but above every other lattice point with x-coordinate between x_i and $x_i + h\Delta$ which had been below L_1. In other words, L_2 is the line connecting $(x_i, y_i - 1/(4h\Delta^2))$ and $(x_i + h\Delta, y_i - h + 1/(2\Delta) - 1/(4h\Delta^2))$. L_2 is the desired boundary for R_i' provided that L_2 crosses the line $y = y_i$ at a point $x = x_i - \epsilon$ for $\epsilon \in (0, 1)$. Where does L_2

hit the line $y = y_i$? We have $\tau/\epsilon = 1/\Delta - 1/(2h\Delta^2)$ so $\epsilon = \tau/(1/\Delta - 1/(2h\Delta^2)) < \tau/(1/(2\Delta)) = 2\Delta\tau = 1/(2h\Delta) < 1$.

To construct $q'_i(x)$ from $q_i(x)$, just use the line of slope γ passing through $(0, q_i(0))$. The set of lattice points on or under that line, between x-coordinates 0 and $h\Delta$, is the same as the set of those on or under $q_i(x)$. However, if $q_i(0)$ is integral, so that $(0, q_i(0))$ is *on* both the original line and the "rotated" one, one may want to raise the line slightly to prevent roundoff errors.

Now we just apply the affine transformation T' which maps $(x, y) \to (x, y')$, where $y' = y + x/\gamma$, to turn Q'_i into a halfspace with a horizontal bounding line and R'_i into the intersection of a halfspace with a horizontal bounding line and a halfspace with a vertical bounding line.

We apply this affine transformation to all points P_j. We need to do orthogonal range search queries in which we need to find the minimum z_j in a translated quadrant or halfspace. However, since z_i is defined only after all $z_1, z_2, ..., z_{i-1}$ are defined, the key values are not known in advance. (The points themselves, however, are known in advance.)

3.1 Running time analysis

Here is what a data structure must support in order to run the algorithm. We are given, in advance, n points P_i in $\mathbb{Z}^2$ with $P_i = (x_i, y_i)$. For each i, we will construct $key(i)$ adaptively in the order $1, 2, 3, ..., n$, as follows. Initialize $key(1)$ in some way. The data structure must be able to execute the following code:

- for $i = 2$ to n do:
  - Find a j minimizing $key(j)$ among those $j < i$ satisfying $x_j \leq x_i$ and $y_j \leq y_i$.
  - Now define $key(i)$ (somehow).
- end for.

The augmented segment tree of Mehlhorn and Näher [5] guarantees the existence of a $O(N \log N \log\log N)$-time algorithm [4]. In fact, a data structure giving a running time of $O(N \log N \log\log N)$ is likely to be implicit in Gabow, Bentley, and Tarjan [2]; however their result as stated (Theorem 3.3 and the discussion above it) is for the case when all $key(i)$ values are known in advance.

We leave open the existence of a $O(N \log N)$-time algorithm, and suggest Willard [6] or Chan, Larsen, and Pătrascu [1] as a possible starting point.

4 Acknowledgments

The authors thank Hal Gabow and Kurt Mehlhorn for their help finding data structures supporting all query and update operations in $O(n \log n \log\log n)$ time.

References

[1] T. M. Chan, K. G. Larsen, and M. Pătrascu. Orthogonal range searching on the RAM, revisited. In F. Hurtado and M. J. van Kreveld, editors, *Symposium on Computational Geometry*, pages 1–10. ACM, 2011.

[2] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *ACM Symposium on Theory of Computing*, pages 135–143, 1984.

[3] L. Golab, H. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1):574–585, 2009.

[4] K. Mehlhorn, 2011. Personal communication.

[5] K. Mehlhorn and S. Näher. Dynamic Fractional Cascading. *Algorithmica*, pages 215–241, 1990.

[6] D. E. Willard. Examining Computational Geometry, Van Emde Boas Trees, and Hashing from the Perspective of the Fusion Tree. *SIAM J. Comput.*, 29(3):1030–1049, 2000.