Deterministic Rectangle Enclosure and Offline Dominance Reporting on the RAM

Peyman Afshani[1]*, Timothy M. Chan[2] **, and Konstantinos Tsakalidis[3]

[1] MADALGO, Department of Computer Science, Aarhus University, Denmark
peyman@madalgo.au.dk
[2] David R. Cheriton School of Computer Science, University of Waterloo, Canada
tmchan@uwaterloo.ca
[3] Computer Science and Engineering Department, CUHK, Hong Kong
tsakalid@cse.cuhk.edu.hk

Abstract. We revisit a classical problem in computational geometry that has been studied since the 1980s: in the *rectangle enclosure* problem we want to report all k enclosing pairs of n input rectangles in 2D. We present the first deterministic algorithm that takes $O(n \log n + k)$ worst-case time and $O(n)$ space in the word-RAM model. This improves previous deterministic algorithms with $O((n \log n + k) \log \log n)$ running time. We achieve the result by derandomizing the algorithm of Chan, Larsen and Pătraşcu [SoCG'11] that attains the same time complexity but in expectation.
The 2D rectangle enclosure problem is related to the *offline dominance range reporting* problem in 4D, and our result leads to the currently fastest deterministic algorithm for offline dominance reporting in any constant dimension $d \geq 4$.
A key tool behind Chan et al.'s previous randomized algorithm is *shallow cuttings for 3D dominance ranges*. Recently, Afshani and Tsakalidis [SODA'14] obtained a deterministic $O(n \log n)$-time algorithm to construct such cuttings. We first present an improved deterministic construction algorithm that runs in $O(n \log \log n)$ time in the word-RAM; this result is of independent interest. Many additional ideas are then incorporated, including a linear-time algorithm for *merging* shallow cuttings and an algorithm for an offline *tree point location* problem.

1 Introduction

We study the problem of *rectangle enclosure*: given a set of n axis-aligned rectangles on the plane, report all k pairs (r_1, r_2) of input rectangles where r_1 completely encloses r_2. This is a classic problem in the field of computational geometry [19] with applications to VLSI design, image processing, computer graphics and databases [21,15,12,6].

* Work supported in part by the Danish National Research Foundation grant DNRF84 through Center for Massive Data Algorithmics (MADALGO)

** The work of this author was done during his visit to the Department of Computer Science and Engineering, Hong Kong University of Science and Technology.

Previous Results. An early paper by Bentley and Wood [3] presented an $O(n \log n + k)$ worst-case time and linear-space algorithm for the related *rectangle intersection* problem (reporting all k pairs (r_1, r_2) where r_1 intersects r_2), raising the question whether the same bound could be achieved for rectangle enclosure. Vaishnavi and Wood [21] first addressed the question presenting an $O(n \log^2 n + k)$-time algorithm that uses $O(n \log^2 n)$ space. Lee and Preparata [15] improved the space bound to linear.

Further improvements were discovered in the 1990s. The linear-space algorithm of Gupta, Janardan, Smid and Dasgupta [12] and an alternative implementation by Lagogiannis, Makris and A. Tsakalidis [14] take $O((n \log n + k) \log \log n)$ worst-case time. Recently, Chan, Larsen and Pătraşcu [6] succeeded in improving the running time to the desired bound of $O(n \log n + k)$ using linear space. However their algorithm uses randomization and thus the time bound holds in expectation. All presented algorithms operate in the word-RAM model with word size $w \geq \log n$. (In fact, for all time bounds that are $\Omega(n \log n)$, they hold in the standard RAM model with $w = \log n$, since we can pre-sort and reduce to rank space.)

It is well-known that the rectangle enclosure problem is reducible to the 4D version of the *offline dominance reporting* problem: given n input and query points in $\mathbb{R}^d$, report the input points that are dominated by each query point ($(p_1, \ldots, p_d)$ is dominated by $(q_1, \ldots, q_d)$ if $p_i < q_i$ for all i). For the reduction it suffices to map each input rectangle $[x_1, x_2] \times [y_1, y_2]$ to a 4D point $(x_1, y_1, -x_2, -y_2)$ and equate the query points with the input points.

Offline dominance reporting is a fundamental problem in the area of orthogonal range searching; it has even found applications outside of computational geometry [4]. Chan, Larsen and Pătraşcu's result implies an algorithm with $O(n \log^{d-3} n + k)$ expected time for any constant dimension $d \geq 4$, where k is the total number of reported points. However their algorithm is randomized. The best deterministic algorithm known requires $O(n \log^{d-2} n + k)$ or $O((n \log^{d-3} n + k) \log \log n)$ time.

Our Contributions. We present the first deterministic algorithm for rectangle enclosure that takes $O(n \log n + k)$ worst-case time and $O(n)$ space in the standard word-RAM model. Our result thus improves over the previous deterministic algorithms of Gupta et al. [12] and Lagogiannis et al. [14] and removes randomization from the algorithm of Chan et al. [6].

Our approach also gives the currently fastest deterministic algorithm for the offline dominance reporting problem for any constant dimension $d \geq 4$, with worst-case running time $O(n \log^{d-3} n + k)$.

Our Approach. Our algorithm may be viewed as a derandomization of Chan, Larsen and Pătraşcu's [6], but significant new ideas are required.

In Chan et al.'s algorithm, randomization was used to construct combinatorial objects that have properties similar to those of *shallow cuttings for 3D dominance ranges.* Shallow cuttings were introduced by Matoušek [18], and a complicated randomized $O(n \log n)$-time algorithm was given by Ramos [20] for constructing shallow cuttings in the more general setting of 3D halfspace ranges. In a recent SODA'14 paper, Afshani and K. Tsakalidis [2] presented the first deterministic $O(n \log n)$-time algorithm for constructing shallow cuttings for 3D dominance ranges using linear space in the pointer machine model. In Section 2 we begin by improving their algorithm to run in $O(n \log \log n)$ worst-case time on the word-RAM. As an immediate consequence we obtain a deterministic algorithm for offline 3D dominance reporting that takes $O(n \log \log n + k)$ worst-case time and linear space in the word-RAM (Section 4); this result is new. Previously only $O(n \log n + k)$ worst-case time could be achieved using linear space [1,17,2].

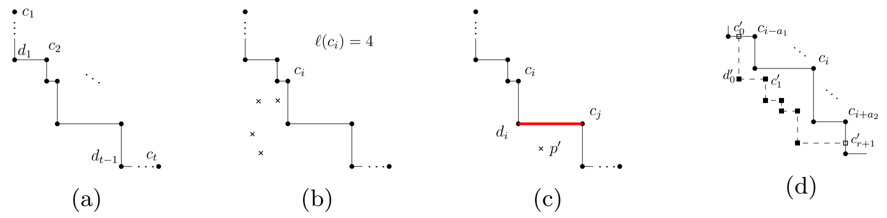
Much further work is needed to derive our result on offline 4D dominance reporting and 2D rectangle enclosure. The crucial new ingredient is an algorithm that can *merge* two shallow cuttings for 3D dominance ranges in *linear* time; this result is obtained by modifying our shallow cutting construction algorithm in interesting ways and is described in Section 3. Then in Section 5 we use an intricate combination of Chan et al.'s approach with the deterministic shallow cutting construction and merging subroutines to achieve our final result. The combination requires a re-organization of the previous algorithm. In particular we isolate a subproblem we call *tree point location* for which we obtain a deterministic algorithm by incorporating planar separators [16] to ideas from Chan et al. This problem may be viewed as a new 2D variant of *fractional cascading* [9] and is thus of independent interest.

Notation and Definitions. Point p *dominates* point q if and only if each coordinate of p is greater than or equal to that of q. To avoid ambiguity for points on the plane we use the term "*covers*" (instead of "dominates"). Let P be a set of n points in $\mathbb{R}^d$. The *level* of any point $p \in \mathbb{R}^d$ (with respect to P) is the number of points in P that are dominated by p. The region of $\mathbb{R}^d$ dominated by p is called a *cell*. The conflict list of a cell is the subset of P that lies inside the cell.

A k-*shallow cutting* for dominance ranges on point set P is a set of *vertices* (points) S such that (i) every vertex in S has level at most $c_{\max}k$ in P for a constant $c_{\max} > 1$ and (ii) any point in $\mathbb{R}^d$ with level in P at most k is dominated by some vertex of S. Shallow cutting S is *optimal* when it contains at most $c_{\max}\frac{n}{k}$ vertices. A planar shallow cutting has the shape of an orthogonal *staircase* curve $c_1 d_1 c_2 \cdots d_{t-1} c_t$ of alternating vertical line segments $c_i d_i = [c_i(x)] \times [c_i(y), d_i(y)]$ and horizontal line segments $d_i c_{i+1} = [d_i(x), c_{i+1}(x)] \times [d_i(y)]$ (Fig. 1a). We call points c_i, d_i, the *corners* of the planar shallow cutting.

2 Construction of 3D Dominance Shallow Cuttings

Theorem 1. *An optimal k-shallow cutting for 3D dominance ranges on n input points and for any integer k can be constructed deterministically in $O(n \log \log n)$ worst-case time and $O(n)$ space.*



Algorithm Sketch. Following [2] we sort the points and sweep a plane parallel to the xy-plane considering the points in non-decreasing z-coordinate. This reduces the problem to the problem of maintaining a 2D shallow cutting under only insertions, specifically the planar shallow cutting of the xy-projection of the points of P that lie *below* the sweep plane. When the point with the next highest z-coordinate is considered (i.e., the next insertion), we update the corresponding 2D shallow cutting. The idea is that we do not need to change the planar shallow cutting for most insertions. However each insertion can increase the level of the shallow cutting corners. Thus, once in a while, the planar shallow cutting needs to be fixed. This is done by removing some consecutive parts of it and then adding new staircase "patches" that are covered by the parts just removed (details will follow). Every time a planar shallow cutting cell is removed, a 3D shallow cutting cell is created using the z-coordinate of the sweep plane (i.e., when a planar staircase corner with coordinates (x, y) is removed, a 3D vertex (x, y, z) is created where z is the coordinate of the sweep plane). Finally when the sweep terminates, the remaining planar shallow cutting cells are turned into 3D shallow cutting cells using $+\infty$ as the z-coordinate. It is easily verified that the produced 3D shallow cutting of P is correct and its size is equal to the number of planar shallow cutting corners removed throughout the algorithm plus the number of planar shallow cutting corners that remain when the sweep terminates.

Remark. The sketched algorithm is a variant of Afshani and Tsakalidis' [2, Section 3] with the significant difference that it is insertion-only (sweeping upwards) as opposed to their deletion-only algorithm (sweeping downwards). Their algorithm has the advantage that it can compute $O(\log n)$ different shallow cuttings in total $O(n \log n)$ time. However a crucial ingredient of that algorithm is an auxiliary data structure ([2, Lemma 2]) that needs to be updated at every sweep point. Unfortunately, we cannot see a way to update the auxiliary data structure in $O(\log \log n)$ time in the word RAM model. Fortunately, as we shall see later, we can achieve the desired $O(n \log \log n)$ running time without any auxiliary data structures, by just changing the direction of the sweep.

The Invariant. The planar shallow cutting is maintained in the form of a staircase $S = c_1 d_1 \cdots c_t$, composed of *inner* corners $d_1, \ldots, d_{t-1}$ and *outer* corners $c_1, \ldots, c_t$. The outer corners c_1 and c_t are (conceptually) at infinity, i.e. $c_1(y) = +\infty$ and $c_t(x) = +\infty$ (Fig. 1a). We maintain the invariant that the inner corners dominate at least k and the outer corners at most $10k$ points.

Details. We now discuss the details of the algorithm. Let $p = (x, y, z)$ be the next point swept by the sweep plane. Remember that we need to insert the point $p' = (x, y)$ into the dynamic planar shallow cutting. To do that we maintain the following structures: first, we use a dynamic van Emde Boas tree on the x-coordinates of the staircase, and second, for every outer corner c of the staircase, we keep track of the number of points it covers, $\ell(c)$ (Fig. 1b), as well as a linked list containing them, $L(c)$ (i.e., the conflict list and its size). Using these we can now insert the point p'. The dynamic van Emde Boas tree enables us to find the inner corner d_i immediately to the left of the point p', helping us decide whether p' lies above or below the staircase (Fig. 1c). In the former case, we are done. In the latter case, for every outer corner c_j that covers p', we increase $\ell(c_j)$ by one and then append p' to $L(c_j)$. If for all such corners we still have $\ell(c_j) \leq 10k$, then the invariant is maintained and thus we are done. However, it is possible that for some corners this invariant is violated. Below we describe how to "patch" such violated invariants.

Complexity. Sorting by z-coordinate takes $O(n \log \log n)$ time in total [13]. Finding d_i takes $O(\log \log n)$ time [22] and thus $O(n \log \log n)$ time in total. It turns out the rest of the algorithm consumes linear time. If there are $m(p')$ corners that cover p', then updating their relevant information takes $O(m(p'))$ time. Note that p' is now added to the conflict lists of $m(p')$ corners. Notice that since the size of each conflict list is $O(k)$, the total running time of this step is $O(Tk)$ where T is size of the shallow cutting. If we can prove that $T = O(n/k)$, then this running time is linear. Now we describe how to maintain the invariant which also guarantees the upper bound on T.

Patching. Let c_i be the leftmost outer corner whose invariant is violated. Let a_1 and a_2 be the largest integers such that all the outer corners $c_{i-a_1}, c_{i-a_1+1}, \ldots, c_{i+a_2}$ have levels greater than $3k$. To patch the staircase we begin by finding a new outer corner c'_0 at the same y-coordinate as c_{i-a_1}, such that it covers $3k$ points, as depicted in Fig. 1d. c'_0 can be found in $O(k)$ time using a linear time selection algorithm on the conflict list of c_{i-a_1}. Next, we find the inner corner d'_0 directly below c'_0 that covers k points. Now, we alternate between finding new outer and inner corners: at the j-th step, we find the outer corner c'_j that dominates $2k$ points, and then the inner corner d'_j that dominates k points. As before, using the right conflict list, each of these corners can be found in $O(k)$ time. The patching is terminated at a point c'_{r+1} with level $3k$ that lies below the outer corner c_{i+a_2} (Fig. 1d). Finally, the new outer and inner corners (from c'_0 to c'_{r+1}) are incorporated into the staircase, the old corners $(c_{i-a_1}, \ldots, c_{i+a_2})$ are removed, and the van Emde Boas tree is also updated to reflect the changes in the staircase.

Analysis of Patching. It easy to see that the overall cost of patching is $O(Tk)$ since each new inner or outer corner can be found in $O(k)$ time. Moreover, the facts that the levels of the removed corners differ from the levels of the created ones by at least k (except for only c'_0 and c'_{r+1}) and that at least 5 corners are created for every patch, suffice to claim that $T \leq C_0 \frac{n}{k}$ for a positive constant C_0. We set $c_{\max} := \max\{C_0, 10\}$. A detailed proof on the symmetric approach is found in [2, Section 3].

3 Merging Two 3D Dominance Shallow Cuttings

We begin by a naive merging algorithm.

Lemma 1. *Consider two point sets P_1 and P_2 that contain n_1 and n_2 points respectively. For $i = 1, 2$, assume we are given a k_i-shallow cutting C_i on P_i of size m_i such that the conflict list of every cell contains at most $\beta_i k_i$ points of P_i. A $(\min\{k_1, k_2\})$-shallow cutting C on the union point set $P = P_1 \cup P_2$ can be built in $O((m_1 + m_2) \log\log(m_1 + m_2))$ time, such that C contains $O(m_1 + m_2)$ cells and the conflict list of every cell in C contains $\leq \beta_1 k_1 + \beta_2 k_2$ input points.*

Proof. Let R be the subset of $\mathbb{R}^3$ that is dominated by at least one vertex in C_1 and at least one vertex in C_2. It is easily seen that R is orthogonally convex and in fact any orthogonal ray to $y = -\infty$ or $x = -\infty$ crosses the boundary of R at most one. Thus, the complexity of the boundary of R is $O(|C_1| + |C_2|) = O(m_1 + m_2)$. Furthermore, the boundary of R can be computed with a straightforward sweep plane algorithm in $O((m_1 + m_2) \log\log(m_1 + m_2))$ time by employing a van Embe Boas tree as the search structure [22]. The shallow cutting C is defined by the vertices of the boundary of R. It is clear that every such vertex dominates either k_1 points from P_1 or k_2 points from P_2 and thus it dominates at least $\min\{k_1, k_2\}$ points of P. Similarly, each vertex on R can dominate at most $\beta_1 k_1$ points of P_1 and $\beta_2 k_2$ points of P_2. $\square$

While the above merging algorithm is quite fast, it worsens the constants behind the parameters of the shallow cutting and thus it cannot be applied more than a constant number of times. In the next theorem, we show how such a “bad” shallow cutting can be refined into an optimal one. For this purpose we also use the following lemma.

Lemma 2. *[5, Theorem 4.3] Online point location queries on a planar orthogonal subdivision of size n can be supported in $O(\log\log n)$ worst-case time and $O(n)$ space.*

Theorem 2. *Let P be a set of n points in $\mathbb{R}^3$ with presorted z-coordinates and let C be a k-shallow cutting on P of size $\alpha n/k$, where the conflict list of every cell has size at most βk, for arbitrary constants $\alpha, \beta > 0$. Then C can be refined into an optimal k'-shallow cutting C' on P in $O(n + \frac{n}{k} \log\log n)$ time, such that $k' = \frac{k}{c_{\max}}$ for a universal constant $c_{\max}$ that does not depend on α and β. Furthermore, C' contains at most $c_{\max} \frac{n}{k'}$ cells and the conflict list of every cell in C' contains at most $c_{\max} k'$ points of P.*

Proof. We build C' using the plane sweep algorithm from the previous section. To review, the algorithm maintains a planar shallow cutting in the form of a staircase S'. To process the next point p, a predecessor query is used to find one corner of the staircase that covers the projection p' of p. As we noted during the proof of Theorem 1, other than this predecessor search, the rest of the algorithm runs in linear time. To remove this bottleneck, we use C to augment each point of P with a correct pointer to a staircase corner that covers it, thus negating the need for the predecessor search.

Hence we project the cutting C on the xy-plane and obtain an orthogonal planar decomposition of disjoint polygonal *regions* in order to support online planar orthogonal point location queries, i.e. report the region that any given query point lies in. Thus, Lemma 2 enables us to perform the following operation in $O(\log\log n)$ time: given a point q in the xy-plane, find the shallow cutting vertex $C(q)$ in C with the largest z-coordinate whose projection covers q.

Observe that the level of every vertex in C' is at most $c_{\max}k' = k$; thus, every vertex of C' is contained in at least one cell of C. We thus maintain one additional invariant in our sweep. Consider a staircase corner $v \in S'$, the shallow cutting vertex $C(v) \in C$ and its conflict list $\ell(C(v))$. We maintain the invariant that if the projection p' of a point $p \in \ell(C(v))$ lies below the staircase S', then p is assigned a pointer to a staircase corner that covers p'. This invariant removes the need for the predecessor search during the sweep.

By looking at the algorithm in Section 2, it is clear that the invariant can only be violated when a new staircase corner v is created on S' (during the patching phase). To fix the invariant, by Lemma 2 we can find the vertex $C(v)$ and its conflict list $\ell(C(v))$. We go through each point in $\ell(C(v))$ and if its projection is covered by v, then we assign it a pointer to v. This takes $O(k)$ time, which is proportional to the time required for creating the staircase corner v. Thus, this incurs only an additive $O(\log\log n)$ time factor per shallow cutting vertex. $\square$

Corollary 1. *Given two point sets $A, B \in \mathbb{R}^3$ presorted by z-coordinate and their respective k-shallow cuttings, for an integer $k = \Omega(\log\log(|A| + |B|))$, an optimal $\frac{k}{c_{\max}}$-shallow cutting on the union point set $A \cup B$ can be constructed deterministically in $O(|A| + |B|)$ worst-case time.*

4 Offline 3D Dominance Reporting

Theorem 3. *Offline 3D dominance reporting on n input points, n query points and k reported points can be solved deterministically in $O(n \log\log n + k)$ worst-case time and $O(n)$ space.*

Proof. We follow the approach of Afshani [1] for online 3D dominance reporting queries in internal memory. We presort all coordinates [13] and construct a single $(\log n)$-shallow cutting for 3D dominance ranges by using the algorithm of Theorem 1 in $O(n \log\log n)$ worst-case time. We *assign* every query point to a cell of the cutting whose vertex dominates it by (offline) point location in a planar orthogonal subdivision obtained from the projection of the cutting. By Lemma 2 this takes $O(n \log\log n)$ worst-case time [5]. We resolve all the assigned queries by solving independently for each of the $O(\frac{n}{\log n})$ cells an offline 3D dominance reporting *subproblem* on the conflict list of the cell. The 3D dominance reporting algorithm of [17] reports all (at most k) output points in $O(\frac{n}{\log n}\log n \log\log n + k) = O(n \log\log n + k)$ total worst-case time. This leaves at most $O(\frac{k}{\log n})$ queries *unresolved*, since each unresolved query reports $\Omega(\log n)$ points. We can also decrease the number of input points to $O(\frac{k}{\log n})$ in the same way by repeating the above with the roles of the input and query points reversed. Finally we solve a single offline 3D dominance reporting problem on all the remaining input points and unresolved queries by the more expensive $O(n \log n + k)$-time algorithm of [17], which now takes $O(k)$ time. $\square$

Corollary 2. *Offline 3D dominance reporting on n input points, n query points and k reported points can be solved deterministically in $O(n+k)$ worst-case time and $O(n)$ space, if $n < \bar{w}^{O(1)}$ and a global look-up table has been constructed in $o(2^{\bar{w}})$ worst-case time for a parameter $\bar{w} \leq w$.*

Proof. We modify the proof of Theorem 3. In the construction of the $(\log n)$-shallow cutting we replace van Emde Boas trees [22] with atomic heaps [10], which decreases the $O(\log\log n)$ search cost to $O(\log_{\bar{w}} n) = O(1)$. The offline planar point location queries can be answered in $O(\log_{\bar{w}} n) = O(1)$ time per query, by a straightforward plane sweep implemented using an atomic heap. The conflict list of each cell has size $O(\log n) = O(\log(\bar{w}^{O(1)})) = o(\bar{w})$. Thus by table lookup, the subproblem on each conflict list can be solved in $O(1)$ time. $\square$

5 Offline 4D Dominance Reporting

A preliminary $O(n \log n \log\log n + k)$ worst-case time and linear-space algorithm for the rectangle enclosure problem is implied by Theorem 3. We obtain a faster deterministic algorithm for rectangle enclosure.

Theorem 4. *Offline 4D dominance reporting problem on n input points, n query points and k reported points can be solved deterministically in $O(n \log n + k)$ worst-case time and $O(n)$ space.*

Algorithm. We follow the approach of Chan, Larsen and Pătraşcu [6, Section 4.3]. We build a complete binary range tree $\mathcal{T}$ over the fourth coordinate of the input points. For each query point, we consider the path from the root of $\mathcal{T}$ to the leaf that contains its successor input point; we associate the query point with the left siblings (if they exist) of the nodes along this path. It then suffices to solve in every node of $\mathcal{T}$ an offline 3D dominance reporting problem between the 3D projections of its input point set and its associated query point set.

To this end, we first *equip* each node at level i of $\mathcal{T}$ with an optimal K_i-shallow cutting for 3D dominance ranges of its input points for some K_i between $\log^2 n$ and $\log^3 n$. At each node every associated query point is *assigned* to a cell of the equipped cutting whose vertex dominates it. All assigned queries are resolved by solving independently for each cell in $\mathcal{T}$ an offline 3D dominance reporting *subproblem* on the cell's conflict list. This leaves at most $O(\frac{k}{\log^2 n})$ queries *unresolved*, since each unresolved query reports $\Omega(K_i) \geq \Omega(\log^2 n)$ points. We can also decrease the number of input points to $O(\frac{k}{\log^2 n})$ in the same way by repeating the above with the roles of the input and query points reversed. Finally we solve a single offline 4D dominance reporting problem on all the remaining input points and unresolved queries.

Complexity. There are $O(\frac{n}{K_i})$ subproblems of size $O(K_i)$ at each level i of $\mathcal{T}$. Hence the total cost $T_{4D}(n, k)$ of the algorithm on n input and query points and k reported points is at most 2 times

$$T_E(n) + T_A(n) + \sum_{i=1}^{\log n} \sum_{j=1}^{O(\frac{n}{K_i})} T_{3D}(O(K_i), k_{ij}) + T_{4D}(O\left(\frac{k}{\log^2 n}\right), k)$$

where (i) $T_E(n)$ represents the cost of equipping the nodes with shallow cuttings, (ii) $T_A(n)$ represents the cost of assigning the queries to cells of the cuttings, (iii) $T_{3D}(O(K_i), k_{ij})$ represents the cost of solving a subproblem on a conflict list of size $O(K_i)$ with output size k_{ij}, and (iv) $T_{4D}(O\left(\frac{k}{\log^2 n}\right), k)$ represents the cost of handling the remaining input points and unresolved queries.

For (i) and (ii), we will show that $T_E(n), T_A(n) = O(n \log n)$. For (iii), we have $T_{3D}(O(K_i), k_{ij}) = O(K_i + k_{ij})$ by applying the algorithm of Corollary 2 with $\bar{w} = \log n$, since $K_i \leq \log^3 n \leq \bar{w}^{O(1)}$. Summing the cost over all i and j yields $O(\sum_{i=1}^{\log n} \frac{n}{K_i} K_i + \sum_{i,j} k_{ij}) = O(n \log n + k)$. For (iv), we can use the more expensive algorithm of [15] with $O(n \log^2 n + k)$ running time, which gives $T_{4D}(O\left(\frac{k}{\log^2 n}\right), k) = O(k)$. The overall running time is thus $O(n \log n + k)$.

Equipping Nodes with Optimal Shallow Cuttings. To show that $T_E(n) = O(n \log n)$, we first construct optimal $(\log^3 n)$-shallow cuttings for the nodes at every level of $\mathcal{T}$ that is a multiple of $\delta \log \log n$, for a constant $\delta > 0$, using the algorithm of Theorem 1. In total this takes $O(n \log \log n \cdot \frac{\log n}{\delta \log \log n}) = O(n \log n)$ time.

For each j that is not a multiple of $\delta \log \log n$, we equip the nodes at level i of $\mathcal{T}$ with an optimal K_i-shallow cutting on its input points in $\mathcal{T}$ by merging the optimal K_{i-1}-shallow cuttings of its two children nodes with the algorithm of Corollary 1. Here, $K_i = \frac{K_{i-1}}{c_{\max}} \Longrightarrow \log^3 n \geq K_i \geq \frac{\log^3 n}{(c_{\max})^{\delta \log \log n}} \geq \log^2 n$ as desired, if we make δ sufficiently small. The entire merging process takes $O(n)$ time per level of $\mathcal{T}$ (since $K_i = \Omega(\log \log n)$) and thus $O(n \log n)$ total time.

Assigning Queries to Cells. To show that $T_A(n) = O(n \log n)$ we formulate a general problem of independent interest.

Problem 1. [Offline 2D Tree Point Location] Given a binary tree where every node contains a planar orthogonal subdivision with rectangular cells, and given a set of query points each of which is associated with a root-to-leaf path in the tree, locate the cell containing q in the subdivisions at all the nodes along the path associated with q, for every query point q.

Lemma 3. *Offline 2D tree point location on n query points and a tree of subdivisions with total size N and tree height $O(\log N)$ can be solved deterministically in $O(N + n \log N)$ worst-case time and $O(n)$ space, assuming pre-sorted coordinates.*

Since the output to the problem consists of $O(\log N)$ cells per query, the above result is optimal. Directly answering each of the $O(n \log N)$ 2D orthogonal point location queries by known results (Lemma 2) would yield a slower $O(N + n \log N \log\log N)$ running time. On the other hand, if the subdivisions in the tree are one-dimensional, then the problem can be solved by the well known technique of *fractional cascading* [9], achieving the same bound as in Lemma 3. Thus, Lemma 3 may be viewed as a generalization of fractional cascading to 2D; no such generalizations were known before (although to be fair our lemma works only in the offline setting).

We will prove Lemma 3 in Section 6, but for now let's see how the tree point location is relevant to our original problem. At each node of $\mathcal{T}$, we form a planar orthogonal subdivision from the projection of the shallow cutting equipped at the node's left sibling. Then assigning queries to cells of the shallow cuttings of their associated nodes is precisely the above tree point location problem. Here, the total size of the planar subdivisions is $N := O\left(\sum_{i=1}^{\log n} \frac{n}{K_i}\right) \le O(\log n \cdot \frac{n}{\log^2 n}) = o(n)$. The tree height is $\log n$, thus by Lemma 3 we get $T_A(n) = O(N + n \log N) = O(n \log n)$.

Remarks. The above algorithm description is actually conceptually cleaner than Chan, Larsen and Pătraşcu's [6], which worked with shallow cuttings of two different ranges of K, namely $K \approx \text{polylog}\, n$ and $K \approx 2^{\sqrt{w}}$. We only need the former, although the expression $2^{\sqrt{w}}$ will appear later in the proof of Lemma 3.

As an immediate corollary to Theorem 4, we can then solve the 2D rectangle enclosure problem in $O(n \log n + k)$ time. Also, we can solve the offline d-dimensional dominance reporting problem in $O(n \log^{d-3} n + k)$ time by a straightforward divide-and-conquer, using the new algorithm for $d = 4$ as the base case.

6 Offline 2D Tree Point Location

To complete the presentation we now prove Lemma 3 and solve the offline 2D tree point location problem. We adapt key ideas from Chan, Larsen and Pătraşcu [6], but in addition incorporate planar separators to get a deterministic algorithm.

Preliminaries. An *r-separator* of a planar graph with n vertices is a subset of $O(\sqrt{rn})$ vertices whose removal yields components of $O(n/r)$ size each. Given d-dimensional input points and query hyper-rectangles (*boxes*), the *offline d-dimensional orthogonal range reporting problem* asks to report the input points that are contained in every query box. Given an orthogonal subdivision of the plane into disjoint rectangles and query points, the *offline 2D orthogonal point location problem* asks report the rectangle that every query point lies in.

Lemma 4. *[11, Theorem 2.2] An r-separator of a planar graph of size n can be computed deterministically in $O(n)$ worst-case time and space.*

Lemma 5. *[6, Lemma 4.2] Offline d-dimensional orthogonal range reporting on n input points, m query boxes and k reported points can be solved deterministically in $O(n \log_b^{d-1} n + b^{d-1} m \log_b^{d-1} n + k)$ worst-case time and $O(n)$ space for a given parameter $b \geq 2$, assuming pre-sorted coordinates.*

Lemma 6. *[6, Lemma 4.1] Offline planar point location for n query points and a orthogonal subdivision of size n can be solved deterministically in $O(n)$ worst-case time and space when $n \leq 2^{O(\sqrt{w})}$, assuming pre-sorted coordinates.*

Lemma 5 follows from a b-ary variant of the standard range tree, while Lemma 6 requires a bit-packing technique of Chan and Pătraşcu [8].

Proof of Lemma 3. First we compute a *sparser* subdivision at every node v of $\mathcal{T}$. Namely, if the subdivision at v has size n_v, we compute a $\left(\frac{n_v}{A}\right)$*-separator* of the subdivision for a parameter $A := 2^{\sqrt{w}}$ with $w = \log N$. This gives $O(\sqrt{n_v \cdot \frac{n_v}{A}}) = O(n_v/\sqrt{A})$ separating rectangles; each remaining hole can be further decomposed into rectangles to yield a subdivision of size $O(n_v/\sqrt{A})$. By Lemma 4 the computation of the separators takes $O(N)$ time total. (This idea of using separators for point location has been used before [5,7].)

Now we locate each query point q in the sparser subdivisions at the nodes along q's path. The key idea from Chan, Larsen and Pătraşcu [6] is to view all these 2D point location queries collectively as a single offline orthogonal range reporting problem in 3D. Namely, we lift each rectangle in the sparser subdivision at node v to a box in 3D, where the range of the box in the third coordinate corresponds to the range of v in the binary tree. This 3D problem involves n points, $O(\frac{N}{\sqrt{A}})$ boxes and output size $k = O(n \log N)$. By Lemma 5 the problem can be solved in $O(n \log_b^2 N + b^2(\frac{N}{\sqrt{A}}) \log_b^2 N + n \log N)$ time. By choosing the parameter $b := A^{1/2-\delta} = 2^{\Theta(\sqrt{\log N})}$, we have $\log_b^2 N = O(\log N)$ and the time bound becomes $O(n \log N)$.

Knowing the cell of the sparser subdivision containing a query point q, we still need to locate the cell of the original subdivision containing q. This reduces to point location in a component, but since each component has size $O(A) = O(2^{\sqrt{w}})$ we can apply the offline point location algorithm of Lemma 6, to finish in time linear to the size of the subdivisions $O(N)$ and the number of queries $O(n \log N)$. ☐

Acknowledgements. We thank Christos Makris from the University of Patras, Greece for preliminary work on this problem.

References

1. Afshani, P.: On dominance reporting in 3D. In: Proc. of the 16th An. ESA. Volume 5193 of LNCS., Springer (2008) 41–51
2. Afshani, P., Tsakalidis, K.: Optimal deterministic shallow cuttings for 3D dominance ranges. In: Proc. of the 25th An. SODA, ACM-SIAM (2014) 1389–1398
3. Bentley, J.L., Wood, D.: An optimal worst case algorithm for reporting intersections of rectangles. IEEE Trans. Computers **29**(7) (1980) 571–577
4. Chan, T.M.: All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. Algorithmica **50**(2) (2008) 236–243
5. Chan, T.M.: Persistent predecessor search and orthogonal point location on the word RAM. ACM Transactions on Algorithms **9**(3) (2013) 22
6. Chan, T.M., Larsen, K.G., Pătraşcu, M.: Orthogonal range searching on the RAM, revisited. In: Proc. of the 27th SoCG, ACM (2011) 1–10
7. Chan, T.M., Pătraşcu, M.: Transdichotomous results in computational geometry, I: Point location in sublogarithmic time. SIAM J. Comp. **39**(2) (2009) 703–729
8. Chan, T.M., Pătraşcu, M.: Counting inversions, offline orthogonal range counting, and related problems. In: Proc. of the 21st An. SODA, ACM-SIAM (2010) 161–173
9. Chazelle, B., Guibas, L.J.: Fractional cascading: I. A data structuring technique. Algorithmica **1**(2) (1986) 133–162
10. Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comp. Syst. Sci. **48**(3) (1994) 533–551
11. Goodrich, M.T.: Planar separators and parallel polygon triangulation. J. Comp. Syst. Sci. **51**(3) (1995) 374–389
12. Gupta, P., Janardan, R., Smid, M., DasGupta, B.: The rectangle enclosure and point-dominance problems revisited. I. J. C. Geom. & Appl. **7**(5) (1997) 437–455
13. Han, Y.: Deterministic sorting in $O(n\log\log n)$ time and linear space. J. Algorithms **50**(1) (2004) 96–105
14. Lagogiannis, G., Makris, C., Tsakalidis, A.K.: A new algorithm for rectangle enclosure reporting. Inf. Process. Lett. **72**(5-6) (1999) 177–182
15. Lee, D.T., Preparata, F.P.: An improved algorithm for the rectangle enclosure problem. J. Algorithms **3**(3) (1982) 218–224
16. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comp. **9**(3) (1980) 615–627
17. Makris, C., Tsakalidis, K.: An improved algorithm for static 3D dominance reporting in the pointer machine. In: Proc. of the 23rd ISAAC. Volume 7676 of LNCS., Springer (2012) 568–577
18. Matoušek, J.: Reporting points in halfspaces. Comp. Geom. **2** (1992) 169–186
19. Preparata, F.P., Shamos, M.I.: Computational Geometry - An Introduction. Springer (1985)
20. Ramos, E.A.: On range reporting, ray shooting and k-level construction. In: Proc. of the 15th SoCG, ACM (1999) 390–399
21. Vaishnavi, V., Wood, D.: Data structures for the rectangle containment and enclosure problems. Comp. Graphics and Image Processing **13**(4) (1980) 372 – 384
22. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Mathematical Systems Theory **10**(1) (1976) 99–127