Dominance Product and High-Dimensional Closest Pair under L_∞*

Omer Gold[1] **and Micha Sharir**[2]

1 **Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel**
omergold@post.tau.ac.il

2 **Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel**
michas@post.tau.ac.il

Abstract

Given a set S of n points in $\mathbb{R}^d$, the Closest Pair problem is to find a pair of distinct points in S at minimum distance. When d is constant, there are efficient algorithms that solve this problem, and fast approximate solutions for general d. However, obtaining an *exact* solution in very high dimensions seems to be much less understood. We consider the high-dimensional L_∞ Closest Pair problem, where $d = n^r$ for some $r > 0$, and the underlying metric is L_∞.

We improve and simplify previous results for L_∞ Closest Pair, showing that it can be solved by a deterministic strongly-polynomial algorithm that runs in $O(DP(n,d)\log n)$ time, and by a randomized algorithm that runs in $O(DP(n,d))$ expected time, where $DP(n,d)$ is the time bound for computing the *dominance product* for n points in $\mathbb{R}^d$. That is a matrix D, such that $D[i,j] = \left|\{k \mid p_i[k] \le p_j[k]\}\right|$; this is the number of coordinates at which p_j dominates p_i. For *integer* coordinates from some interval $[-M, M]$, we obtain an algorithm that runs in $\tilde{O}\left(\min\{Mn^{\omega(1,r,1)}, DP(n,d)\}\right)$ time[1], where $\omega(1,r,1)$ is the exponent of multiplying an $n \times n^r$ matrix by an $n^r \times n$ matrix.

We also give slightly better bounds for $DP(n,d)$, by using more recent rectangular matrix multiplication bounds. Computing the dominance product itself is an important task, since it is applied in many algorithms as a major black-box ingredient, such as algorithms for APBP (all pairs bottleneck paths), and variants of APSP (all pairs shortest paths).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Closest Pair, Dominance Product, L_∞ Proximity

Digital Object Identifier 10.4230/LIPIcs...1

1 Introduction

Finding the closest pair among a set of n points in $\mathbb{R}^d$ was among the first studied algorithmic geometric problems, considered at the origins of computational geometry; see [20, 18]. The distance between pairs of points is often measured by the L_τ metric, for some $1 \le \tau \le \infty$, under which the distance between the points $p_i = (p_i[1], \ldots, p_i[d])$ and $p_j = (p_j[1], \ldots, p_j[d])$

* Work on this paper has been supported by Grant 892/13 from the Israel Science Foundation, by Grant 2012/229 from the U.S.-Israeli Binational Science Foundation, by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

[1] The $\tilde{O}(\cdot)$ notation hides poly-logarithmic factors.

is $\mathrm{dist}_\tau(p_i,p_j) = \|p_i - p_j\|_\tau = \left(\sum_{k=1}^{d} \left|p_i[k] - p_j[k]\right|^\tau\right)^{1/\tau}$, for $\tau < \infty$, and $\mathrm{dist}_\infty(p_i,p_j) = \|p_i - p_j\|_\infty = \max_k \left|p_i[k] - p_j[k]\right|$, for $\tau = \infty$. The Closest Pair problem and its corresponding *decision* variant, under the L_τ-metric, are defined as follows.

Closest Pair: Given a set S of n points in $\mathbb{R}^d$, find a pair of distinct points $p_i, p_j \in S$ such that $\mathrm{dist}_\tau(p_i,p_j) = \min_{\ell \neq m}\{\mathrm{dist}_\tau(p_\ell,p_m) \mid p_\ell, p_m \in S\}$.

Closest Pair Decision: Given a set S of n points in $\mathbb{R}^d$, and a parameter $\delta > 0$, determine whether there is a pair of distinct points $p_i, p_j \in S$ such that $\mathrm{dist}_\tau(p_i,p_j) \leq \delta$.

Throughout the paper, the notation L_τ Closest Pair refers to the Closest Pair problem under some *specific* metric L_τ, for $1 \leq \tau \leq \infty$ (and we will mostly consider the case $\tau = \infty$).

In the *algebraic computation tree model* (see [3]), the Closest Pair problem has a complexity lower bound of $\Omega(n \log n)$ (for any L_τ metric), even for the one-dimensional case $d = 1$, as implied from a lower bound for the Element-Uniqueness problem [3].

As for upper bounds, Bentley and Shamos [5, 4] were the first who gave a deterministic algorithm for finding the closest pair under the L_2 metric that runs in $O(n \log n)$ time for any *constant* dimension $d \geq 1$, which is optimal in the algebraic computation tree model, for any fixed d. Their algorithm uses the *divide-and-conquer* paradigm, and became since, a classical textbook example for this technique. In 1976 Rabin presented, in a seminal paper [19], a *randomized* algorithm that finds the closest pair in $O(n)$ expected time, using the *floor* function (which is not included in the algebraic computation tree model). His algorithm uses random sampling to decompose the problem into smaller subproblems, and uses the floor function in solving them, for a total cost of $O(n)$ expected time. Later, in 1979, Fortune and Hopcroft [9] gave a deterministic algorithm that uses the floor function, and runs in $O(n \log\log n)$ time.

The bounds above hold as long as the dimension d is constant, as they involve factors that are exponential in d. Thus, when d is large (e.g., $d = n$), the problem seems to be much less understood. Shamos and Bentley [5] conjectured in 1976 that, for $d = n$, and under the L_2 metric, the problem can be solved in $O(n^2 \log n)$ time; so far, their conjectured bound is considerably far from the $O(n^\omega)$ state-of-the-art time bound for this case [13], where $\omega < 2.373$ denotes the exponent for matrix multiplication (see below). If one settles on approximate solutions, many efficient algorithms were developed in the last two decades, mostly based on LSH (locality sensitive hashing) schemes, and dimensionality reduction via the Johnson-Lindenstrauss transform; see [2, 1] for examples of such algorithms. These algorithms are often used for finding *approximate nearest neighbors*, which itself is of major importance and in massive use in many practical fields of computer science. Nevertheless, finding an *exact* solution seems to be a much harder task.

We consider the case where d depends on n, i.e., $d = n^r$ for some $r > 0$. Note that a naive brute-force algorithm runs in $O(n^2 d)$ time and works for any metric L_τ. For some L_τ metrics, a much faster solution is known; see [13]. Specifically, the L_2 Closest Pair problem can be solved by one algebraic matrix multiplication, so for example when $d = n$, it can be solved in $O(n^\omega)$ time (as already mentioned above). If $\tau \geq 2$ is an *even* integer, then L_τ Closest Pair can be solved in $O(\tau n^\omega)$ time. However, for other L_τ metrics, such as when τ is *odd* (or fractional), or the L_∞ metric, the known solutions are significantly inferior.

For the L_1 and L_∞ metrics, Indyk *et al.* [13] obtained the first (and best known until now) non-naive algorithms for the case $d = n$. For L_1, they gave an algorithm that runs in $O\left(n^{\frac{\omega+3}{2}}\right) = O(n^{2.687})$ time, and for L_∞, one that runs in $O\left(n^{\frac{\omega+3}{2}} \log D\right) = O(n^{2.687} \log D)$ time, where D is the diameter of the given point-set. The bound for L_∞ is *weakly polynomial*,

due to the dependence on D, and, for real data, only yields an approximation. Their paper is perhaps the most related to our work.

Our new approach is based on two main observations. The first is showing a reduction from L_∞ Closest Pair Decision to another well-studied problem, *dominance product*. The second is by showing we can solve the optimization problem deterministically by executing the decision procedure only $O(\log n)$ times.

We also give improved runtime analysis for the dominance product problem, defined as follows.

Dominance Product: given a set S of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, compute a matrix D such that for each $i, j \in [n]$, $D[i,j] = \Big|\{k \mid p_i[k] \leq p_j[k]\}\Big|$.

This matrix is called the *dominance product* or *dominance matrix* for S. For $d = n$, there is a non-trivial strongly subcubic algorithm by Matoušek [17] (see Section 4), and a slightly improved one by Yuster [23]. For $d \leq n$, there are extensions of Matoušek's algorithm by Vassilevska-Williams, Williams, and Yuster [21]. All of them use fast matrix multiplications.

Dominance product computations were liberally used to improve some fundamental algorithmic problems. For example, Vassilevska-Williams, Williams, and Yuster [21], give the first strongly subcubic algorithm for the *all pairs bottleneck paths* (APBP) problem, using dominance product computations. Duan and Pettie [8] later improved their algorithm, also by using dominance product computations, in fact, their time bound for (max, min)-product match the current time bound of computing the dominance product of n points in $\mathbb{R}^n$. Yuster [23] showed that APSP can be solved in strongly subcubic time if the number of distinct weights of edges emanating from any fixed vertex is $O(n^{0.338})$. In his algorithm, he uses dominance product computation as a black box.

1.1 Preliminaries

We review some notations that we will use throughout the paper. We denote by $[N] = \{1, \ldots, \lceil N \rceil\}$, the set of the first $\lceil N \rceil$ natural numbers succeeding zero, for any $N \in \mathbb{R}^+$. For a point $p \in \mathbb{R}^d$, we denote by $p[k]$ the k-th coordinate of p, for $k \in [d]$. For a matrix A, we denote the *transpose* of A by A^T. The $\tilde{O}(\cdot)$ notation hides poly-logarithmic factors.

Most of the algorithms discussed in this paper heavily rely on fast matrix multiplication algorithms. Throughout the paper, $\omega < 2.373$ denotes the exponent of multiplying two $n \times n$ matrices [22, 14], and $\omega(1, r, 1)$ refers to the exponent of multiplying an $n \times n^r$ matrix by an $n^r \times n$ matrix, for some $r > 0$; see [12, 15]. For more details on rectangular matrix multiplication exponents, we refer the reader to the seminal work of Huang and Pan [12], and to a more recent work of Le Gall [15, 16].

1.2 Our Results

Let $DP(n, d)$ denote the runtime order for computing the dominance product (defined above) of n points in $\mathbb{R}^d$. We obtain the following results for the L_∞ Closest Pair problem in $\mathbb{R}^d$, where $d = n^r$, for some $r > 0$.

▶ **Theorem 1.** L_∞ Closest Pair *can be solved by a deterministic algorithm that runs in* $O(DP(n, d) \log n)$ *time.*

Theorem 1 improves the $O(n^{2.687} \log D)$ time bound of Indyk *et al.* [13] (see above) in two aspects, first is that the polynomial factor $n^{2.687}$ goes slightly down to $DP(n, n) = n^{2.684}$, which we then improve further to $n^{2.6598}$ in Theorem 4; this holds also for Theorem 2, stated

below. The second aspect is that the $\log D$ factor is replaced by a $\log n$ factor, which makes our algorithm strongly-polynomial, independent of the diameter of the given point-set.

For the proof of Theorem 1, we first show a reduction from L_∞ Closest Pair Decision to dominance product computation, then we show that the optimization problem can be cleverly solved deterministically by executing the decision procedure only $O(\log n)$ times.

▶ **Theorem 2.** L_∞ Closest Pair *can be solved by a randomized algorithm that runs in* $O(DP(n,d))$ *expected time.*

▶ **Theorem 3.** *For points with integer coordinates from* $[-M,M]$, L_∞ Closest Pair *can be solved by a deterministic algorithm that runs in* $\tilde{O}\left(\min\{Mn^{\omega(1,r,1)}, DP(n,d)\}\right)$ *time.*

From Theorem 3 we obtain that for n points in $\mathbb{R}^n$ with small integer coordinates we can solve the *optimization* problem in $O(n^\omega)$ time, which is a significant improvement compared to the general case from Theorems 1 and 2.

Additionally, in Theorem 4 we give improved bounds for $DP(n,d)$.

▶ **Theorem 4.** *given a set* S *of* n *points* $p_1,\ldots,p_n$ *in* $\mathbb{R}^d$*, their dominance product can be computed in* $O(DP(n,d))$ *time, where*

$$DP(n,d) \leq \begin{cases} d^{0.697}n^{1.896}+n^{2+o(1)} & \text{if } d \leq n^{\frac{\omega-1}{2}} \leq n^{0.687} \\ d^{0.909}n^{1.75} & \text{if } n^{0.687} \leq d \leq n^{0.87} \\ d^{0.921}n^{1.739} & \text{if } n^{0.87} \leq d \leq n^{0.963} \\ d^{0.931}n^{1.73} & \text{if } n^{0.963} \leq d \leq n^{1.056} \end{cases}$$

In particular, we obtain that $DP(n,n)=n^{2.6598}$, which improves Yuster's $O(n^{2.684})$ time bound. As mentioned above, these bounds will slightly improve the time bounds for algorithms that use dominance product computation as a bottleneck step (see some examples above). In the rest of the paper we will often refer to the function $DP(n,d)$ above.

2 L_∞ Closest Pair

Recall that, given a set S of n points $p_1,\ldots,p_n$ in $\mathbb{R}^d$, the L_∞ Closest Pair problem is to find a pair of points (p_i,p_j), such that $i\neq j$ and $\|p_i-p_j\|_\infty = \min_{\ell\neq m\in[n]}\|p_\ell-p_m\|_\infty$. The corresponding decision version of this problem is to determine whether there is a pair of distinct points (p_i,p_j) such that $\|p_i-p_j\|_\infty \leq \delta$, for a given $\delta>0$.

Naively, we can compute all the distances between every pair of points in $O(n^2d)$ time, and choose the smallest one. However, as we see next, a significant improvement can be achieved, for any $d=n^r$, for any $r>0$.

Specifically, we first obtain the following theorem.

▶ **Theorem 5.** *Given a parameter* $\delta>0$*, and a set* S *of* n *points* $p_1,\ldots,p_n$ *in* $\mathbb{R}^d$*, the set of all pairs* (p_i,p_j) *with* $\|p_i-p_j\|_\infty \leq \delta$*, can be computed in* $O(DP(n,d))$ *time.*

Proof. First, we note the following trivial but useful observation.

▶ Observation 6. For a pair of points $p_i,p_j\in\mathbb{R}^d$, $\|p_i-p_j\|_\infty\leq\delta \iff p_i[k]\leq p_j[k]+\delta$ and $p_j[k]\leq p_i[k]+\delta$, for every coordinate $k\in[d]$.

Indeed, a pair of points (p_i,p_j) satisfies $\|p_i-p_j\|_\infty = \max_{k\in[d]}|p_i[k]-p_j[k]| \leq \delta \iff$ for every coordinate $k\in[d]$, $|p_i[k]-p_j[k]|\leq\delta$. The last inequalities hold iff $p_i[k]-p_j[k]\leq\delta$ and $p_j[k]-p_i[k]\leq\delta$, or, equivalently, iff $p_i[k]\leq p_j[k]+\delta$ and $p_j[k]\leq p_i[k]+\delta$, for each

$k \in [d]$. Although the rephrasing in the observation is trivial, it is crucial for our next step. It can be regarded as a (simple) variant of what is usually referred to as "Fredman's trick" (see [11]).

For every $i \in [n]$ we create a new point $p_{n+i} = p_i + (\delta, \delta, \ldots, \delta)$. Thus in total, we now have $2n$ points. Concretely, for every $i \in [n]$, we have the points

$$\begin{array}{rcccccc} p_i & = \big(& p_i[1], & p_i[2], & \ldots, & p_i[d] & \big), \\ p_{n+i} & = \big(& p_i[1]+\delta, & p_i[2]+\delta, & \ldots, & p_i[d]+\delta & \big). \end{array}$$

We compute the dominance matrix D_δ for these $2n$ points, using the algorithm from Section 4.1. By Observation 6, a pair of points (p_i, p_j) satisfies

$$\|p_i - p_j\|_\infty \le \delta \iff (D_\delta[i, n+j] = d) \ \wedge \ (D_\delta[j, n+i] = d)\,,$$

so we can find all these pairs in $O(n^2)$ additional time. Clearly, the runtime is determined by the time bound of computing the dominance matrix D_δ, that is, $O(DP(n,d))$. ◀

The proof of Theorem 5 shows that solving the L_∞ Closest Pair Decision is not harder than computing the dominance matrix for n points in $\mathbb{R}^d$. In particular, by the decision tree complexity bound for computing dominance matrices, as discussed in Section 4, the following result is straightforward.

▶ **Corollary 7.** *Given a parameter $\delta > 0$, and a set S of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, determining all pairs $i \neq j$ such that $\|p_i - p_j\|_\infty \le \delta$ can be done using $O(dn \log n)$ pairwise comparisons (of real numbers).*

By Corollary 7, we obtain that the 2-linear decision tree complexity for the L_∞ Closest Pair Decision problem is $O(dn \log n)$. This bound matches a special case of an old conjectured algorithmic complexity bound by Shamos and Bentley (see Section 1, and [5]).

2.1 Solving the Optimization Problem

The algorithm from Theorem 5 solves the L_∞ Closest Pair Decision problem. It actually gives a stronger result, as it finds *all* pairs of points (p_i, p_j) such that $\|p_i - p_j\|_\infty \le \delta$. We use this algorithm in order to solve the optimization problem L_∞ Closest Pair.

As a "quick and dirty" solution, one can solve the optimization problem by using the algorithm from Theorem 5 to guide a binary search over the diameter W of the input point set, which is at most twice the largest absolute value of the coordinates of the input points. If the coordinates are integers then we need to invoke the algorithm from Theorem 5 $O(\log W)$ times. If the coordinates are reals, we invoke it $O(B)$ times for B bits of precision. However, the dependence on W makes this method weakly polynomial, and, for real data, only yields an approximation. As we show next, this naive approach can be replaced by strongly-polynomial algorithms, A deterministic one that runs in $O(DP(n,d) \log n)$ time, and a randomized one that runs in $O(DP(n,d))$ expected time.

Deterministic strongly-polynomial algorithm.

▶ **Theorem 8.** *Given a set S of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, the L_∞ Closest Pair problem can be solved for S in $O(DP(n,d) \log n)$ time.*

Proof. Since the distance between the closest pair of points, say p_i, p_j, is

$$\delta_0 = \|p_i - p_j\|_\infty = \max_{k \in [d]} \big|p_i[k] - p_j[k]\big|,$$

it is one of the $O(n^2 d)$ values $p_\ell[k] - p_m[k]$, $\ell, m \in [n]$, $k \in [d]$. Our goal is to somehow search through these values, using the decision procedure (i.e., the algorithm from Theorem 5). However, enumerating all these values takes $\Omega(n^2 d)$ time, which is too expensive, and pointless anyway, since by having them, the closest pair can be found immediately. Instead, we proceed in the following more efficient manner.

For each $k \in [d]$, we sort the points of S in increasing order of their k-th coordinate. This takes $O(nd \log n)$ time in total. Let $\left(p_1^{(k)}, \ldots, p_n^{(k)}\right)$ denote the sequence of the points of S sorted in increasing order of their k-th coordinate. For each k, let $M^{(k)}$ be an $n \times n$ matrix, so that for $i, j \in [n]$, we have

$$M^{(k)}[i, j] = p_i^{(k)}[k] - p_j^{(k)}[k].$$

We are in fact interested only in the upper triangular portion of $M^{(k)}$, where its elements are positive, but for simplicity of presentation, we ignore this issue. (We view the row indices from bottom to top, i.e., the first row is the bottommost one, and the column indices from left to right.)

Observe that each row of $M^{(k)}$ is sorted in decreasing order and each column is sorted in increasing order. Under these conditions, the selection algorithm of Frederickson and Johnson [10] can find the t-largest element of $M^{(k)}$, for any $1 \le t \le n^2$, in $O(n)$ time.[2] (Note that we do not need to explicitly construct the matrices $M^{(k)}$, this will be too expensive. The bound of Frederickson-Johnson's algorithm holds as long as each entry of $M^{(k)}$ is accessible in $O(1)$ time, like in our case.)

We use this method to conduct a simultaneous binary search over all d matrices $M^{(k)}$ to find δ_0. At each step of the search we maintain two counters $L_k \le H_k$, for each k. Initially $L_k = 1$ and $H_k = n^2$. The invariant that we maintain is that, at each step, δ_0 lies in between the L_k-th and the H_k-th largest elements of $M^{(k)}$, for each k.

Each binary search step is performed as follows. We compute $r_k = \lfloor (L_k + H_k)/2 \rfloor$, for each k, and apply the Frederickson-Johnson algorithm to retrieve the r_k-th largest element of $M^{(k)}$, which we denote as δ_k, in total time $O(nd)$. We give δ_k the weight $H_k - L_k + 1$, and compute the weighted median δ_{med} of $\{\delta_1, \ldots, \delta_d\}$. We run the L_∞ Closest Pair Decision procedure of Theorem 5 on δ_{med}. Suppose that it determines that $\delta_0 \le \delta_{\text{med}}$. Then for each k for which $\delta_k \ge \delta_{\text{med}}$ we know that $\delta_0 \le \delta_k$, so we set $H_k := r_k$ and leave L_k unchanged. Symmetric actions are taken if $\delta_0 > \delta_{\text{med}}$. In either case, we remove roughly one quarter of the candidate differences; that is, the sum $\sum_{k \in [d]} (H_k - L_k + 1)$ decreases by roughly a factor of 3/4. Hence, after $O(\log n)$ steps, the sum becomes $O(d)$, and a straightforward binary search through the remaining values finds δ_0. The overall running time is

$$O(nd \log n + DP(n, d)(\log n + \log d)).$$

Since in our setting d is polynomial in n, and $nd \ll DP(n, d)$, we obtain that the overall runtime is $O(DP(n, d) \log n)$. This completes the proof of Theorem 1. ◀

Randomized algorithm. Using randomization, we can improve the time bound of the preceding deterministic algorithm to equal the time bound of computing the dominance product $O(DP(n, d))$ in expectation. This can be done by using a randomized optimization technique by Chan [6]. Among the problems for which this technique can be applied, Chan specifically addresses the Closest Pair problem.

[2] Simpler algorithms can select the t-largest element in such cases in $O(n \log n)$ time, which is also sufficient for our approach.

▶ **Theorem 9** (Chan [6]). *Let U be a collection of objects. If the* Closest Pair Decision *problem can be solved in $O(T(n))$ time, for an arbitrary distance function $d : U \times U \to \mathbb{R}$, then the* Closest Pair *problem can be solved in $O(T(n))$ expected time, assuming that $T(n)/n$ is monotone increasing.*

We refer the reader to [6], for the proof of Theorem 9. By Theorem 5, L_∞ Closest Pair Decision can be solved in $O(DP(n,d))$ time. Clearly, $DP(n,d)/n$ is monotone increasing in n. Hence, by Theorem 9, we obtain a randomized algorithm for L_∞ Closest Pair that runs in $O(DP(n,d))$ expected time, as stated in Theorem 2.

3 L_∞ Closest Pair with Integer Coordinates

A considerable part of the algorithm from the previous section is the reduction to computing a suitable dominance matrix. The algorithms for computing dominance matrices given in Section 4 do not make any assumptions on the coordinates of the points, and support real numbers. When the coordinates are bounded integers, we can improve the algorithms. In particular, for n points in $\mathbb{R}^n$ with small integer coordinates we can solve the *optimization* problem in $O(n^\omega)$ time, which is a significant improvement compared to the $O(n^{2.6598})$ time bound of our previous algorithm for this case[3]. Our improvement is based on techniques for computing $(\min,+)$-matrix multiplication over integer-valued matrices.

▶ **Theorem 10.** *Let S be a set of n points $p_1,\ldots,p_n$ in $\mathbb{R}^d$ such that $d = n^r$ for some $r > 0$, and for all $i \in [n]$, $k \in [d]$, $p_i[k]$ is an integer in $[-M,M]$. Then the L_∞ closest pair can be computed in*

$$\tilde{O}\left(\min\left\{Mn^{\omega(1,r,1)},\ DP(n,d)\right\}\right) \text{ time.}$$

We first define $(\max,+)$-product and $(\min,+)$-product over matrices.

▶ **Definition 11** (Distance products of matrices). Let A be an $n \times m$ matrix and B be an $m \times n$ matrix. The $(\max,+)$-product of A and B, denoted by $A \star B$, is the $n \times n$ matrix C whose elements are given by

$$c_{ij} = \max_{1 \le k \le m} \{a_{ik} + b_{kj}\}, \quad \text{for } i,j \in [n].$$

Similarly, the $(\min,+)$-product of A and B denoted by $A * B$ is the $n \times n$ matrix C' whose elements are given by

$$c'_{ij} = \min_{1 \le k \le m} \{a_{ik} + b_{kj}\}, \quad \text{for } i,j \in [n].$$

We refer to either of the $(\min,+)$-product or the $(\max,+)$-product as a *distance product*.

The distance product of an $n \times m$ matrix by an $m \times n$ matrix can be computed naively in $O(n^2 m)$ time. When $m = n$, the problem is equivalent to APSP (all pairs shortest paths) problem in a directed graph with real edge weights, and the fastest algorithm known is a recent one by Chan and Williams [7] that runs in $O\left(n^3/2^{\sqrt{\Omega(\log n)}}\right)$ time. It is a prominent long-standing open problem whether a truly subcubic algorithm for this problem exists. However, when the entries of the matrices are integers, we can convert distance products of matrices into standard algebraic products. We use a technique by Zwick [24].

[3] For integer coordinates that are bounded by a constant, the L_∞-diameter of the points is also a constant (bounded by twice the largest coordinate), hence, one can use the decision procedure to (naively) guide a binary search over the diameter in constant time.

▶ **Lemma 12** (Zwick [24]). *Given an $n \times m$ matrix $A = \{a_{ij}\}$ and an $m \times n$ matrix $B = \{b_{ij}\}$ such that $m = n^r$ for some $r > 0$, and all the elements of both matrices are integers from $[-M, M]$, their $(\min, +)$-product $C = A * B$ can be computed in $\tilde{O}(Mn^{\omega(1,r,1)})$ time.*

With minor appropriate modifications, the $(\max, +)$-product of matrices A and B can be computed within the same time as in Lemma 12.

We now give an algorithm for computing all-pairs L_∞ distances, by using the fast algorithm for computing $(\max, +)$-product over bounded integers.

▶ **Lemma 13.** *Let S be a set of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$ such that $d = n^r$ for some $r > 0$, and for all $i \in [n]$, $p_i[k]$ is an integer from the interval $[-M, M]$, for all $k \in [d]$. Then the L_∞ distances between all pairs of points (p_i, p_j) from S can be computed in $\tilde{O}(Mn^{\omega(1,r,1)})$ time.*

Proof. We create the $n \times d$ matrix $A = \{a_{ik}\}$ and the $d \times n$ matrix $B = (-A)^T = \{b_{ki}\}$, where

$$a_{ik} = p_i[k], \quad \text{for } i \in [n],\ k \in [d]$$

$$b_{ki} = -p_i[k], \quad \text{for } i \in [n],\ k \in [d].$$

Now we compute the $(\max, +)$-product $C = A \star B$. The matrix L of all-pairs L_∞-distances is then easily seen to be

$$L[i, j] = \max\{C[i, j], C[j, i]\} = \|p_i - p_j\|_\infty ,$$

for every pair $i, j \in [n]$.

Clearly, the runtime is determined by computing the $(\max, +)$-product $C = A \star B$. This is done as explained earlier, and achieves the required running time. ◀

Consequently, by taking the minimum from the algorithm above, and the (say, deterministic) algorithm from Section 2, we obtain that for points in $\mathbb{R}^d$ with integer coordinates from $[-M, M]$, where $d = n^r$ for some $r > 0$, we can find the L_∞ closest pair in

$$\tilde{O}\left(\min\left\{Mn^{\omega(1,r,1)},\ DP(n, d)\right\}\right) \text{ time,}$$

as stated in Theorem 3.

4 Dominance Products

We recall the dominance product problem: given n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, we want to compute a matrix D such that for each $i, j \in [n]$,

$$D[i, j] = \Big|\{k \mid p_i[k] \le p_j[k]\}\Big|.$$

It is easy to see that the matrix D can be computed naively in $O(dn^2)$ time. Note that, in terms of decision tree complexity, it is straightforward to show that $O(dn \log n)$ pairwise comparisons suffice for computing the dominance product of n points in $\mathbb{R}^d$. However, the actual best known time bound to solve this problem is significantly larger than its decision tree complexity bound.

The first who gave a truly subcubic algorithm to compute the dominance product of n points in $\mathbb{R}^n$ is Matoušek [17]. We first outline his algorithm, and then present our extension and improved runtime analysis.

▶ **Theorem 14** (Matoušek [17]). *Given a set S of n points in $\mathbb{R}^n$, the dominance matrix for S can be computed in $O(n^{\frac{3+\omega}{2}}) = O(n^{2.687})$ time.*

Proof. For each $j \in [n]$, sort the n points by their j-th coordinate. This takes a total of $O(n^2 \log n)$ time. Define the *j-th rank of point p_i*, denoted as $r_j(p_i)$, to be the position of p_i in the sorted list for coordinate j. Let $s \in [\log n, n]$ be a parameter to be determined later. Define n/s pairs (assuming for simplicity that n/s is an integer) of $n \times n$ Boolean matrices $(A_1, B_1), \ldots, (A_{n/s}, B_{n/s})$ as follows:

$$A_k[i,j] = \begin{cases} 1 & \text{if } r_j(p_i) \in [ks, ks+s) \\ 0 & \text{otherwise,} \end{cases} \qquad B_k[i,j] = \begin{cases} 1 & \text{if } r_j(p_i) \geq ks+s \\ 0 & \text{otherwise,} \end{cases}$$

for $i, j \in [n]$. Put $C_k = A_k \cdot B_k^T$. Then $C_k[i,j]$ equals the number of coordinates t such that $r_t(p_i) \in [ks, ks+s)$, and $r_t(p_j) \geq ks+s$.

Thus, by letting $C = \sum_{k=1}^{n/s} C_k$, we have that $C[i,j]$ is the number of coordinates t such that $p_i[t] \leq p_j[t]$ and $\lfloor r_t(p_i)/s \rfloor < \lfloor r_t(p_j)/s \rfloor$.

Next, we compute a matrix E such that $E[i,j]$ is the number of coordinates t such that $p_i[t] \leq p_j[t]$ and $\lfloor r_t(p_i)/s \rfloor = \lfloor r_t(p_j)/s \rfloor$. Then $D := C + E$ is the desired dominance matrix.

To compute E, we use the n sorted lists we computed earlier. For each pair $(i,j) \in [n] \times [n]$, we retrieve $q := r_j(p_i)$. By reading off the adjacent points that precede p_i in the j-th sorted list in reverse order (i.e., the points at positions $q-1$, $q-2$, etc.), and stopping as soon as we reach a point p_k such that $\lfloor r_j(p_k)/s \rfloor < \lfloor r_j(p_i)/s \rfloor$, we obtain the list $p_{i_1}, \ldots, p_{i_l}$ of $l \leq s$ points such that $p_{i_x}[j] \leq p_i[j]$ and $\lfloor r_j(p_i)/s \rfloor = \lfloor r_j(p_{i_x})/s \rfloor$. For each $x = 1, \ldots, l$, we add a 1 to $E[i_x, i]$. Assuming constant time lookups and constant time probes into a matrix (as is standard in the real RAM model), this entire process takes only $O(n^2 s)$ time. The runtime of the above procedure is therefore $O(n^2 s + \frac{n}{s} \cdot n^\omega)$. Choosing $s = n^{\frac{\omega-1}{2}}$, the time bound becomes $O(n^{\frac{3+\omega}{2}})$. ◀

Yuster [23] has slightly improved this algorithm to run in $O(n^{2.684})$ time, by using rectangular matrix multiplication.

4.1 Generalized and Improved Bounds

We extend Yuster's idea to obtain bounds for dimension $d = n^r$, for the entire range $r > 0$, and, at the same time, give an improved time analysis, using the recent bounds for rectangular matrix multiplications of Le Gall [15, 16] coupled with an interpolation technique. This analysis is not trivial, as Le Gall's bounds for $\omega(1, r, 1)$ are obtained by a nonlinear optimization problem, and are only provided for a few selected values of r (see Table 1 in [16] and [15]). Combining Le Gall's exponents with an interpolation technique, similar to the one used by Huang and Pan [12], we obtain improved bounds for all values $d = n^r$, for any $r > 0$.

Note that the matrices A_k and B_k, defined above, are now $n \times d$ matrices. Thus, the sum C defined earlier, can be viewed as a product of block matrices

$$C = \begin{bmatrix} A_1 & A_2 & \cdots & A_{n/s} \end{bmatrix} \cdot \begin{bmatrix} B_1^T \\ B_2^T \\ \vdots \\ B_{n/s}^T \end{bmatrix}.$$

Thus, to compute C we need to multiply an $n \times (dn/s)$ matrix by a $(dn/s) \times n$ matrix. Computing E in this case can be done exactly as in Matoušek's algorithm, in $O(nds)$ time.

r	ω	ζ
$r_0 = 1.0$	$\omega_0 = 2.372864$	$\zeta_0 = 0.6865$
$r_1 = 1.1$	$\omega_1 = 2.456151$	$\zeta_1 = 0.7781$
$r_2 = 1.2$	$\omega_2 = 2.539392$	$\zeta_2 = 0.8697$
$r_3 = 1.3$	$\omega_3 = 2.624703$	$\zeta_3 = 0.9624$
$r_4 = 1.4$	$\omega_4 = 2.711707$	$\zeta_4 = 1.0559$

Table 1 The relevant entries from Le Gall's table. The dominance product can be computed in $O(n^{\omega_i})$ time, for dimension $d_i = n^{\zeta_i}$.

Consider first the case where d is small; concretely, $d \leq n^{\frac{\omega-1}{2}}$. In this case we compute C using the following result by Huang and Pan.

▶ **Lemma 15** (Huang and Pan [12]). *Let $\alpha = \sup\{0 \leq r \leq 1 \mid w(1,r,1) = 2 + o(1)\}$. Then for all $n^\alpha \leq m \leq n$, one can multiply an $n \times m$ matrix with an $m \times n$ matrix in time $O\left(m^{\frac{\omega-2}{1-\alpha}} n^{\frac{2-\omega\alpha}{1-\alpha}}\right)$.*

Huang and Pan [12] showed that $\alpha > 0.294$. Recently, Le Gall [15, 16] improved the bound on α to $\alpha > 0.302$. By plugging this into Lemma 15, we obtain that multiplying an $n \times m$ matrix with an $m \times n$ matrix, where $n^\alpha \leq m \leq n$, can be done in time $O(m^{0.535} n^{1.839})$.

From the above, computing C and E can be done in $O\left((dn/s)^{0.535} n^{1.839} + dns\right)$ time. By choosing $s = n^{0.896}/d^{0.303}$, the runtime is asymptotically minimized, and we obtain the time bound $O(d^{0.697} n^{1.896})$. This time bound holds only when $n^\alpha < n^{0.302} \leq dn/s \leq n$, which yields the time bound

$$O(d^{0.697} n^{1.896} + n^{2+o(1)}), \text{ for } d \leq n^{(\omega-1)/2} \leq n^{0.687}.$$

We now handle the case $d > n^{(\omega-1)/2}$. Note that in this case, $dn/s > n$ (for s as above), thus, we cannot use the bound from Lemma 15. Le Gall [15, 16] gives a table (Table 1 in [16] and [15]) of values r (he refers to them as k), including values of $r > 1$ (which is what we need), with various respective exponents $\omega(1, r, 1)$. We will confine ourselves to the given bounds for the values $r_1 = 1.1$, $r_2 = 1.2$, $r_3 = 1.3$, and $r_4 = 1.4$. We denote their corresponding exponents $\omega(1, r_i, 1)$ by $\omega_1 \leq 2.456151, \omega_2 \leq 2.539392$, $\omega_3 \leq 2.624703$, and $\omega_4 \leq 2.711707$ respectively. The exponent for $r_0 = 1$ is $\omega_0 = \omega \leq 2.372864$ (see [22, 14]).

The algorithm consists of two parts. For a parameter s, that we will fix shortly, the cost of computing $C = A \cdot B^T$ is $O\left(n^{\omega_r}\right)$, where ω_r is a shorthand notation for $\omega(1, r, 1)$, and where $n^r = dn/s$, and the cost of computing E is $O(nds) = O\left(s^2 n^r\right)$. Dropping the constants of proportionality, and equating the two expressions, we choose

$$s = n^{(\omega_r - r)/2}, \qquad \text{that is,} \qquad d = sn^{r-1} = n^{(\omega_r + r)/2 - 1} = n^{\zeta_r},$$

for $\zeta_r = (\omega_r + r)/2 - 1$. Put $\zeta_i = \zeta_{r_i}$, for the values $r_0, \ldots, r_4$ mentioned earlier; see Table 1.

Now if we are lucky and $d = n^{\zeta_i}$, for $i = 0, 1, 2, 3, 4$, then the overall cost of the algorithm is $O(n^{\omega_i})$. For in-between values of d, we need to interpolate, using the following bound, which is derived in the earlier studies (see, e.g., Huang and Pan [12]), and which asserts that, for $a \leq r \leq b$, we have

$$\omega_r \leq \frac{(b-r)\omega_a + (r-a)\omega_b}{b-a}. \tag{1}$$

$\zeta_{\min}$	$\zeta_{\max}$	u	v
0.687	0.87	0.909	1.75
0.87	0.963	0.921	1.739
0.963	1.056	0.931	1.73

Table 2 The time bound for computing dominance product for n points in dimension $n^{\zeta_{\min}} \leq d \leq n^{\zeta_{\max}}$ is $O(d^u n^v)$.

That is, given $d = n^\zeta$, where $\zeta_i \leq \zeta \leq \zeta_{i+1}$, for some $i \in \{0, 1, 2, 3\}$, the cost of the algorithm will be $O(n^{\omega_r})$, where r satisfies

$$\zeta = \zeta_r = \frac{\omega_r + r}{2} - 1.$$

Substituting the bound for ω_r from (1), with $a = r_i$ and $b = r_{i+1}$, we have

$$\frac{(r_{i+1} - r)\omega_i + (r - r_i)\omega_{i+1}}{r_{i+1} - r_i} + r = 2(\zeta + 1).$$

Eliminating r, we get

$$r = \frac{2(\zeta + 1)(r_{i+1} - r_i) - r_{i+1}w_i + r_i w_{i+1}}{w_{i+1} + r_{i+1} - w_i - r_i}, \tag{2}$$

and the cost of the algorithm will be $O(n^{\omega_r})$, where

$$\omega_r \leq \frac{(r_{i+1} - r)\omega_i + (r - r_i)\omega_{i+1}}{r_{i+1} - r_i}. \tag{3}$$

Note that r is a linear function of ζ, and so is ω_r. Writing $\omega_r = u\zeta + v$, the cost is

$$O(n^{\omega_r}) = O\left(n^{u\zeta+v}\right) = O(d^u n^v).$$

The values of u and v for each of our intervals are given in Table 2. (The first row covers the two intervals $1.0 \leq r \leq 1.1$ and $1.1 \leq r \leq 1.2$, as the bounds happen to coincide there.) See also (**??**) in Section 1.2. We have provided explicit expressions for $DP(n, d)$ only for $d \leq n^{\zeta_4} = n^{1.056}$, which includes the range $d \leq n$, which is the range one expects in practice. Nevertheless, the recipe that we provide can also be applied to larger values of d, using larger entries from Le Gall's table [15, 16]. Dropping constant factors, we denote the time bound for computing the dominance product of n points in $\mathbb{R}^d$ by $DP(n, d)$; see Theorem 4 in Section 1.2. by plugging the corresponding values of $0.302 < r < 1$ from Le Gall's Table 1 in [16]. We also note that, for $d = n$, the time bound is $O(n^{2.6598})$, which improves Yuster's $O(n^{2.684})$ time bound mentioned above.

References

1 Nir Ailon and Bernard Chazelle. The fast Johnson-Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009.

2 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

3 Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proc. of the 15th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 80–86, 1983.

4 Jon Louis Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

5 Jon Louis Bentley and Michael Ian Shamos. Divide-and-conquer in multidimensional space. In *Proc. of the 8th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 220–230, 1976.

6 T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22(4):547–567, 1999.

7 Timothy M. Chan and Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In *Proc. of the 27th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 1246–1255, 2016.

8 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proc. of the 20th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 384–391, 2009.

9 Steve Fortune and John Hopcroft. A note on Rabin's nearest-neighbor algorithm. *Inform. Process. Lett.*, 8(1):20–23, 1979.

10 Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in x + y and matrices with sorted columns. *Journal of Computer and System Sciences*, 24(2):197 – 208, 1982.

11 M. L. Fredman. How good is the information theory bound in sorting? *Theoret. Comput. Sci*, 1(4):355–361, 1976.

12 Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.

13 Piotr Indyk, Moshe Lewenstein, Ohad Lipsky, and Ely Porat. Closest pair problems in very high dimensions. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 782–792, 2004.

14 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th International Sympos. on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

15 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proc. 53rd Annu. IEEE Sympos. on Foundations of Computer Science (FOCS)*, pages 514–523, 2012.

16 François Le Gall. Faster algorithms for rectangular matrix multiplication. *CoRR*, abs/1204.1111, 2012.

17 Jiří Matoušek. Computing dominances in E^n. *Inform. Process. Lett.*, 38(5):277–278, 1991.

18 Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, NY, 1985.

19 Michael Rabin. Probabilistic algorithms. In *Algorithms and Complexity, Recent Results and New Directions*, Academic Press, pages 21–39, 1976.

20 Michael Ian Shamos. Geometric complexity. In *Proc. of 7th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 224–233, 1975.

21 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(1):173–189, 2009.

22 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Sympos. on Theory of Computing (STOC)*, pages 887–898, 2012.

23 Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *Proc. 20th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 950–957, 2009.

24 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.