An Optimized Divide-and-Conquer Algorithm for the Closest-Pair Problem in the Planar Case

José C. Pereira  
DEEI-FCT  
Universidade do Algarve  
Campus de Gambelas  
8005-139 Faro, Portugal  
unidadeimaginaria@gmail.com

Fernando G. Lobo  
CENSE and DEEI-FCT  
Universidade do Algarve  
Campus de Gambelas  
8005-139 Faro, Portugal  
fernando.lobo@gmail.com

Abstract

We present an engineered version of the divide-and-conquer algorithm for finding the closest pair of points, within a given set of points in the XY-plane. In this version of the algorithm, only two pairwise comparisons are required in the combine step, for each point that lies in the \(2\delta\)-wide vertical slab. The correctness of the algorithm is shown for all Minkowski distances with \(p \geq 1\). We also show empirically that, although the time complexity of the algorithm is still \(O(n \lg n)\), the reduction in the total number of comparisons leads to a significant reduction in the total execution time, for inputs with size sufficiently large.

1 Introduction

The Closest-Pair problem is considered an “easy” Closest-Point problem, in the sense that there are a number of other geometric problems (e.g. nearest neighbors and minimal spanning trees) that find the closest pair as part of their solution [Bentley, 1980, p. 226]. This problem and its generalizations arise in areas such as statistics, pattern recognition and molecular biology.

At present time, many algorithms are known for solving the Closest-Pair problem in any dimension \(k \geq 2\), with optimal time complexity (see Smid [2000] for an overview of Closest-Point problem algorithms and generalizations). The Closest-Pair is also one of the first non-trivial computational problems that was solved efficiently using the divide-and-conquer strategy and it became since a classical, textbook example for this technique.

In this paper we consider only algorithms for the Closest-Pair problem that can be implemented in the algebraic computation tree model. For this model any algorithm has time complexity \(\Omega(n \lg n)\). With more powerful machine models, where randomization, the floor function, and indirect addressing are available, faster algorithms can be designed [Smid, 2000].
Historical Background

An algorithm with optimal time complexity $O(n \lg n)$ for solving the Closest-Pair problem in the planar case appeared for the first time in 1975, in a computational geometry classic paper by Shamos [1975]. This algorithm was based on the Voronoi polygons.

The first optimal algorithm for solving the Closest-Pair problem in any dimension $k \geq 2$ is due to Bentley and Shamos [1976]. Using a divide-and-conquer approach to initially solve the problem in the plane, those authors were able to generalize the planar process to higher dimensions by exploring a sparsity condition induced over the set of points in the $k$-plane.

For the planar case, the original procedure and other versions of the divide-and-conquer algorithm usually compute at least seven pairwise comparisons for each point in the central slab, within the combine step (see Kleinberg and Tardos [2005], Bentley and Shamos [1976], and Cormen et al. [2001], for instance).

In 1998, Zhou, Xiong, and Zhu presented an improved version of the planar procedure, where at most four pairwise comparisons need to be considered in the combine step, for each point lying on the left side (alternatively, on the right side) of the central slab. In the same article, Zhou et al. introduced the “complexity of computing distances”, which measures “the number of Euclidean distances to compute by a closest-pair algorithm” [Jiang and Gillespie, 2007]. The core idea behind this definition is that, since the Euclidean distance is usually more expensive than other basic operations, it may be possible to achieve significant efficiency improvements by reducing this complexity measure.

More recently, Ge, Wang, and Zhu used some sophisticated geometric arguments to show that it is always possible to discard one of the four pairwise comparisons in the combine step, thus reducing significantly the complexity of computing distances, and presented their enhanced version of the Closest-Pair algorithm, accordingly [Ge et al., 2006].

In 2007, Jiang and Gillespie presented another version of the Closest-Pair divide-and-conquer algorithm which reduced the complexity of computing distances by a logarithmic factor. However, after performing some algorithmic experimentation, the authors found that, albeit this reduction, the new algorithm was “the slowest among the four algorithms” [Jiang and Gillespie, 2007] that were included in the comparative study. The experimental results also showed that the fastest among the four algorithms was in fact a procedure named Basic-2, where two pairwise comparisons are required in the combine step, for each point that lies in the central slab and, therefore, has a relative high complexity of computing distances. The authors conclude that the simpler design in the combine step, and a consequent correct balance in trading expensive operations with cheaper ones are the main factors for explaining the success of the Basic-2 algorithm.

In this paper we present a detailed version of the Basic-2 algorithm. We show that only two pairwise comparisons are required in the combine step, for each point that lies in the central slab, and that this number of comparisons is minimal. This result and the subsequent correctness of the Basic-2 algorithm is shown for all Minkowski distances with $p \geq 1.$

1 According to Bentley [1980], Shamos attributes the discovery of this procedure to H.R. Strong.
2 The article in question was published in Chinese. See Ge et al. [2006] and Jiang and Gillespie [2007] for some explicit references.
3 In fairness to all parts involved, we must say that all the main results presented in this paper, including the design of the Basic-2 algorithm, were obtained in a completely independent fashion from any previous work by other authors. It was only during the process of putting our ideas in to writing that we came across the articles by Zhou et al. Ge et al.
The rest of the paper is organized as follows. In Section 2, we review the classic Closest-Pair algorithm as presented by Bentley and Shamos [1976]. In Section 3, we present our detailed version of the Basic-2 algorithm and give the correspondent proof of correctness. In Section 4, we present a comparative empirical study between the classic Closest-Pair and the Basic-2 algorithm, and discuss the experimental results obtained with distinct Minkowski distances.

2 The divide-and-conquer algorithm in the plane

The following algorithm for solving the planar version of the Closest-Pair problem was first presented by Bentley and Shamos [1976].

Let \( P \) be a set of \( n \geq 2 \) points in the XY-plane. The closest pair in \( P \) can be found in \( O(n \lg n) \) time using the divide-and-conquer algorithm shown in Figure 1.

\[
\text{Closest-Pair}(P) \\
1\text{ Presort points in } P \text{ along the } x\text{-coordinate.} \\
2\text{ Split the ordered set } P \text{ into two equal-sized subsets by the vertical line } l, \text{ defined by the equation } x = x_{\text{median}}. \\
3\text{ Solve the problem recursively in the left and right subsets. This will give the left-side and right-side minimal distances } d_L \text{ and } d_R, \text{ respectively.} \\
4\text{ Find the minimal distance } d_{LR} \text{ among the pair of points in which one point lies on the left of the dividing vertical line and the other point lies to the right.} \\
5\text{ The final answer is the minimum between } d_L, d_R, \text{ and } d_{LR}. \tag{a}
\]

\( \text{Figure 1: Pseudocode for the divide-and-conquer Closest-Pair algorithm, first presented by } \text{Bentley and Shamos } [1976]. \)

Since we are splitting a set of \( n \) points in two sets of \( n/2 \) points each, the recurrence relation describing the running time of the Closest-Pair algorithm is \( T(n) = 2T(n/2) + f(n) \), where \( f(n) \) is the running time for finding the distance \( d_{LR} \) in step 4.

At first sight it seems that something of the order of \( n^2/4 \) distance comparisons will be required to compute \( d_{LR} \). However, Bentley and Shamos [1976] noted that the knowledge of both distances \( d_L \) and \( d_R \) induces a sparsity condition over the set \( P \).

Let \( \delta = \min(d_L, d_R) \) and consider the vertical slab of width \( 2\delta \) centered at line \( l \). If there is any pair in \( P \) closer than \( \delta \), both points of the pair must lie on opposite sides within the slab. Also, because the minimum separation distance of points on either side of \( l \) is \( \delta \), any square region of the slab, with side \( 2\delta \), “can contain at most a constant number \( c \) of points” [Bentley and Shamos, 1976], depending on the used metric.

As a consequence of this sparsity condition, if the points in \( P \) are presorted by y-coordinate, the computation of \( d_{LR} \) can be done in linear time. Therefore, we obtain the recurrence relation \( T(n) = 2T(n/2) + O(n) \), giving an \( O(n \lg n) \) asymptotically optimal algorithm.

[1976], and Jiang and Gillespie [2007] which, obviously, take precedence and deserve due credit.

\footnote{In the original article, Bentley and Shamos [1976], used the Minkowski distance \( d_\infty \) and obtained the value \( c = 12 \).}
The Basic-2 algorithm

In this Section we discuss a detailed version of the Basic-2 algorithm, which was first presented by Jiang and Gillespie [2007].

The Basic-2 algorithm is an optimized version of the Bentley and Shamos procedure for the planar case discussed in section [2]. In fact, the Basic-2 algorithm is the same as the CLOSEST-PAIR algorithm (see Figure [1]), with the sole difference that the computation of the distance \( d_{LR} \) in step 4 now requires only two pairwise comparisons per point to find the closest pair within the central slab. The pseudocode for computing the \( d_{LR} \) distance in the Basic-2 algorithm is shown in Figure 3.

The time complexity of the BASIC-2.S4 algorithm is obviously \( O(n) \), since it traverses once the arrays \( Y_L \) and \( Y_R \) in a “hopscotch” manner (see Figure [2]), and it takes constant time on each iteration.

Since we are only interested in performing step 4 of the Closest-Pair algorithm, in the following we assume that \( x_{\text{median}}, d_L, \) and \( d_R \) are already computed. We also assume that the array \( Y \) contains a \( y \)-sorted partition of all points in \( P \), i.e., the first and second halves of \( Y \) contain the points that are to the left and to the right of \( l \), respectively, and both halves are sorted along the \( y \)-coordinate.

We denote the vertical slab centered at line \( x = x_{\text{median}} \) of width \( 2\delta \) by the symbol \( \Delta \) and the central line by \( l \).

Before we show the correctness of the algorithm we first prove the following

---

5The structure of the array \( Y \) may seem difficult to obtain, and therefore, be an extra source of complexity in the overall algorithm. However, this is not the case because the structure arises as a natural consequence from the need to maintain the \( y \)-presorting throughout the recursive calls in \( O(n) \) time.
Lemma 1. Let \( d_p : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R} \) denote the Minkowski \( p \)-distance, \( 1 \leq p \leq \infty \). Let \( P_0 = (x_0, y_0) \in Y_L \) (respectively, \( Y_R \)) be an arbitrary point lying in the central slab \( \Delta \), and let \( Y_0 \subseteq Y_R \) (respectively, \( Y_L \)) be the array of points that lie opposite and above \( P_0 \), sorted along the \( y \)-coordinate. The closest point to \( P_0 \), in respect to \( d_p \), is either the first or the second element of \( Y_0 \).

Proof. We first give proof for the Minkowski distance for \( p = 1 \), defined by \( d_1(A, B) = |x_A - x_B| + |y_A - y_B| \). We assume, without loss of generality, that \( P_0 = (0, 0) \in Y_L \) and, as a consequence, that \( Y_0 \subseteq Y_R \). Let \( A = (a, b) \), \( 0 \leq a \leq \delta \), \( b \geq 0 \) be the first point in \( Y_0 \). We note that, because \( Y_0 \) is sorted along the \( y \)-coordinate, it is sufficient to consider the case where \( b = 0 \), since all others cases with \( b > 0 \) can be obtained by making an upper translation. This translation does not disrupts the relative positions of the elements in \( Y_0 \) and therefore, all arguments presented for \( b = 0 \) will remain valid. So, let \( A = (a, 0) \) and let \( P = (x, y) \), \( 0 \leq x \leq \delta \), \( y \geq 0 \), be any other point in \( Y_0 \). We must consider three cases (illustrated in Figures 4, 5, and 6).
Case 1: $a = \frac{\delta}{2}$. We find that
\[
0 \leq x \leq \delta \iff \\
\iff 0 - \frac{\delta}{2} \leq x - \frac{\delta}{2} \leq \delta - \frac{\delta}{2} \iff \\
\iff -\frac{\delta}{2} \leq x - \frac{\delta}{2} \leq \frac{\delta}{2} \iff \\
\iff |x - \frac{\delta}{2}| \leq \frac{\delta}{2}
\]
On the other hand, we have
\[
d_1(A, P) \geq \delta \iff \\
\iff |x - \frac{\delta}{2}| + y \geq \delta \iff \\
\iff y \geq \delta - |x - \frac{\delta}{2}| \geq \delta - \frac{\delta}{2} = \frac{\delta}{2}
\]
Therefore,
\[
d_1(P_0, P) = x + y \geq x + \frac{\delta}{2} \geq \frac{\delta}{2} = d_1(P_0, A)
\]
which is to say that $A$ is a closest point to $P_0$, in $Y_0$.
We note that, if we take $y = \frac{\delta}{2}$, the first three points in $Y_0$ may have coordinates $A = (\frac{\delta}{2}, 0); \ P = (\delta, \frac{\delta}{2}); \ Q = (0, \frac{\delta}{2})$, respectively. This is the limit-case depicted in Figure 4, where $A$ and $Q$ are both the closest points to $P_0$, in $Y_0$. 

Figure 4: Possible location for the first three points, $A$, $P$, and $Q$ in Case 1. When $A$ is centered it is possible for both $A$ and $Q$ to be the closest points to $P_0$, in $Y_0$. However, this is a limit-case.
Case 2: $0 \leq a < \frac{\delta}{2}$. We consider two possibilities

i) Let $y > \frac{\delta}{2}$, then

$$d_1(P_0, P) = x + y > x + \frac{\delta}{2} \geq \frac{\delta}{2} > a = d_1(P_0, A).$$

ii) Let $y \leq \frac{\delta}{2}$, then

$$d_1(A, P) \geq \delta \iff |x - a| + y \geq \delta \iff$$

$$\iff |x - a| \geq \delta - y \geq \delta - \frac{\delta}{2} = \frac{\delta}{2} \iff$$

$$\iff x - a \geq \frac{\delta}{2} \vee x - a \leq -\frac{\delta}{2} \iff$$

$$\iff x \geq a + \frac{\delta}{2} \vee x \leq a - \frac{\delta}{2} < 0 \Rightarrow$$

$$\Rightarrow x \geq a + \frac{\delta}{2} \quad \text{Contradiction}$$

Therefore,

$$d_1(P_0, P) = x + y \geq a + \frac{\delta}{2} + y \geq \frac{\delta}{2} > a = d_1(P_0, A).$$

Considering i) and ii) we conclude that $A$ is the closest point to $P_0$ in $Y_0$. 

---

Figure 5: Possible location for the first three points, $A$, $P$, and $Q$ in Case 2. When $A$ is close to $P_0$ all other points in $Y_0$ are “pushed” away by the sparsity of $\Delta$. 

- $P_0 \in Y_L$
- $A$
- $Q$
- $\delta$
- $\delta$
- $Y_L \leftarrow Y_R$
Case 3: $\frac{\delta}{2} < a \leq \delta$. We must consider two possibilities

i) Let $x \geq a$, then

$$d_1(P_0, P) = x + y \geq a + y \geq a = d_1(P_0, A).$$

and, as in the previous cases, $A$ is the closest point to $P_0$, in $Y_0$.

ii) Let $x < a$, then

$$d_1(A, P) \geq \delta \iff a - x + y \geq \delta \iff y \geq \delta - a + x$$

This means that it is possible to have at least one point $P = (x, y) \in Y_0$ such that $d_1(P_0, P) = x + y < a = d_1(P_0, A)$, and $0 \leq x < a$, $\delta - a + x \leq y \leq a - x$. Let $Q = (x_1, y_1) \in Y_0$, and assume that $y_1 \leq y$ (i.e., $Q$ precedes $P$ in $Y_0$). We know that

$$d_1(A, Q) \geq \delta \iff |x_1 - a| + y_1 \geq \delta \iff$$

$$\iff x_1 \geq a + \delta - y_1 \vee x_1 \leq a - \delta + y_1 \quad (1)$$

From the first inequality of (1) we find that

$$x_1 \geq a + \delta - y_1 \geq a + \delta - y > x + y + \delta - y = \delta + x \geq \delta$$

and this means that, in this case, for $Q$ to precede $P$ in $Y_0$, it must lie outside the $\Delta$ slab, which is a contradiction.

The second inequality of (1) holds only if we choose $\delta - a \leq y_1 \leq y$ to guarantee that $0 \leq x_1 \leq y_1 - (\delta - a)$. However, for this choice of the $Q$ coordinates, and from the
sparsity of $\Delta$ we get

\[
\begin{align*}
d_1(P,Q) &\geq \delta \iff |x_1 - x| + y - y_1 \geq \delta \\
&\iff x_1 - x \geq \delta + y_1 - y \\
&\quad \vee \quad x_1 - x \leq -\delta + y - y_1 \\
&\iff x_1 \geq x + \delta + y_1 - (\delta - a + x) \\
&\quad \vee \quad x_1 \leq (x + y) - \delta - y_1 \\
&\iff x_1 \geq y_1 + a \\
&\quad \vee \quad x_1 < a - \delta - y_1 \\
&\iff x_1 > y_1 + a - \delta \\
&\quad \vee \quad x_1 < a - \delta \leq 0 \quad (2)
\end{align*}
\]

The first inequality of (2) is a contradiction with our current working hypothesis (i.e., with the second inequality of (1)) and the second inequality of (2) implies that $Q \not\in Y_0$. So, if there is one point $P \in Y_0$ closer to $P_0$ than point $A$, then no other point, but $A$, can precede $P$ in $Y_0$. Now, suppose that there is another point $Q \in Y_0$ that is also closer to $P_0$ than point $A$. This means that, as with $P$, no other point can precede $Q$ in $Y_0$. However, since both $P$ and $Q$ are in $Y_0$, one must precede the other, which is a contradiction. Therefore, in this case, the only point possibly closer to $P_0$ than point $A$ is the second element of $Y_0$.

From all the previous three cases we may conclude that the closest point to $P_0$, in $Y_0$, is either the first or the second element of $Y_0$. This proves Lemma 1 for the Minkowski distance $d_1$.

Figure 7: Unit circles for various Minkowski $p$-distances.

To obtain proof for all other Minkowski distances $d_p$, $p > 1$, we take into account the fact that the convex neighborhoods generated by the Minkowski distances possess a very straightforward order relation, where larger values of $p$ correspond to larger unit circles, as shown in Figure 7. This ordering means that the sparsity effect within the $\Delta$ slab will be similar, but somewhat stronger, for larger values of $p$. Therefore, the precedent analysis of $d_1$ not only remains valid for $p > 1$ but, in a sense, the corresponding geometric relations between elements of $Y_0$ are expected to be more “tight” for all other Minkowski distances.

The preceding Lemma 1 establishes an upper bound of two for the number of pairwise comparisons needed to be computed for each point in the $\Delta$ slab. Also, the analysis made in Case 3 of the corresponding proof shows that this bound is tight, i.e., we have established the following

**Corollary 2.** For each point in the central slab $\Delta$, the minimum number of pairwise comparisons required to compute $d_{LR}$, in the divide-and-conquer CLOSEST-PAIR algorithm, is two.
Correctness

To prove the correctness of the B ASIC-2.S4 algorithm we consider the following

Loop Invariant. At the start of each iteration of the main loop, left and right are references to the points of $Y_L$ and $Y_R$, respectively, with minimum y-coordinates, that still need to be compared with points on the opposite side. Also, $d_{\text{min}}$ corresponds to the value of the minimum distance found among all pairs of points previously checked.

We show that all three properties Initialization, Maintenance, and Termination hold for this loop invariant.

Initialization At the start of the first iteration, the loop invariant holds since left and right are references to the first elements of $Y_L$ and $Y_R$, and both arrays are y-sorted in ascending order (by construction). Also, no pair with points on opposite sides of line $l$ was checked yet, so the current minimum distance is the minimum between the left-side and right-side minimal distances $d_{\text{Lmin}}$ and $d_{\text{Rmin}}$, respectively. This value is stored in $d_{\text{min}}$.

Maintenance Assuming that the loop invariant holds for all previous iterations we now enter the next iteration. The first thing the loop does is to compute the distance between the points referenced by left and right and, if that is the case, it updates the value of $d_{\text{min}}$. Next, the loop determines which of the two points has smaller y-coordinate. Let us assume, without loss of generality, that in this iteration left is the lowest point. Since left $\in Y_L$, the algorithm checks to see if there is at least one more point in $Y_R$ (denoted by next[$Y_R$]), following the point right. If there is such point, the algorithm computes the distance between left and next[$Y_R$], and updates the value of $d_{\text{min}}$ accordingly.

By our hypothesis we know that right and next[$Y_R$] are the points on the right-hand side with the smallest y-coordinates that are still greater, at most equal, to left’s y-coordinate. Therefore, and taking into account Lemma 1, we conclude that the value of $d_{\text{min}}$ corresponds to the minimum between the previous minimal distance and the minimum distance computed for all pairs of points that contain the left point.

The iteration ends by incrementing the reference left to the next point in $Y_L$, which means that the original left point will no longer be available for comparison. Note also that the right reference remains the same so that the corresponding point will still be compared with other points on future iterations (see Figure 8). We may conclude that the new pair of references left and right, and the new value $d_{\text{min}}$ still satisfy the loop invariant.

Termination The loop ends when one of the references, left or right, reaches the end of the corresponding array, $Y_L$ or $Y_R$, respectively. Let us assume, without loss of generality, that the left reference reaches the end of the array $Y_L$ and terminates the loop. This means that it was the left reference that was incremented at the last iteration and so, it was this reference that corresponded to the lowest point. Accordingly, the loop computed the distances between left and the two closer, but not lower, points in $Y_R$ and updated $d_{\text{min}}$. As a consequence, all remaining pairs of points are composed by the point left and points that belong to the array $Y_R$ and lie in higher, more distant positions. Therefore, we have computed all distances between pairs of opposite points that may lie at a distance smaller than the current minimal distance, and so the value $d_{\text{min}}$ corresponds to the minimal distance between all pairs of points in $\Delta$. 

10
Figure 8: Some iterations of the main loop in Basic-2.S4. (a) Compute distance between left and right, the first elements of $Y_L$ and $Y_R$, respectively. (b)-(c) Point left is lower than point right so, compute distance between left and the next element in $Y_R$. Update left to the next element in $Y_L$. Compute distance between left and right. (d)-(e) Point right is lower than point left so, compute distance between right and the next element in $Y_L$. Update right to the next element in $Y_R$. Compute distance between left and right.

The Basic-2.S4 algorithm is correct and, therefore, the Basic-2 algorithm is also correct, for any Minkowski distance with $p \geq 1$.

4 Empirical Time Analysis

The Basic-2 algorithm has been implemented and tested with randomly generated inputs, starting with 125 thousand points and doubling the input size until 16 million points. For each input size, 50 independent executions were performed. For each specific generated input, we applied the Basic-2 algorithm as well as the standard divide-and-conquer algorithm, described in Cormen et al. [2001], which computes seven pairwise comparison for each point in the central slab. We refer to the standard algorithm as the Basic-7, following the naming convention presented in Jiang and Gillespie [2007].

Both algorithms were implemented in the C programming language. The source code is avail-
The recursion stopped for a number of points less or equal to 10. The algorithms were tested with Minkowski distances \( d_1, d_2, d_{3,1415} \) and \( d_\infty \). For each independent run, we measured the execution time of the entire algorithm. The results are summarized in Figure 9.

![Figure 9: Running time ratios of Basic-2 over Basic-7 for various Minkowski distances.](image)

As expected, and in accordance with the results obtained by Jiang and Gillespie [2007], our simulation shows that Basic-2 is faster than the standard divide-and-conquer algorithm, Basic-7. Although BASIC-2.S4 introduces a few extra relational comparisons (see pseudocode in Figure 3), these are negligible compared to the savings that occur due to the reduction in the total number of distance function calls.

The Basic-2 algorithm is about 20% faster than the standard divide and conquer algorithm for the larger input sizes, when using the Minkowski distances \( d_1 \) and \( d_\infty \). The speedup is more pronounced for the case when using the Minkowski distance \( d_2 \), with Basic-2 being nearly 36% faster for the larger input sizes. The reason for the greater speedup when using Minkowski distance \( d_2 \) is because the computation of the distance function is more expensive in this case, and therefore, savings in distance function calls have a more profound effect on the overall execution time of the algorithm. This fact is fully confirmed by the results obtained when using the somewhat exotic distance \( d_{3,1415} \). Due to the non-integer value of \( p \), the cost of computing this kind of distance function is highly inflated and so it is possible to observe speedups over 100% for the Basic-2 algorithm.
5 Final Remarks

In this paper we analyzed the Basic-2 algorithm, which is an optimized version of the Bentley and Shamos procedure for the planar case where the computation of the distance $d_{LR}$ requires only two pairwise comparisons per point to find the closest pair within the central slab. The Basic-2 algorithm was first presented by Jiang and Gillespie [2007].

In Corollary 2 we proved that the two comparisons per point are a minimum for the divide-and-conquer CLOSEST-PAIR algorithm, for all Minkowski distances with $p \geq 1$. This result is a direct consequence of the strength of the sparsity condition, which is induced over the set of points in the plane by the knowledge of $d_L$ and $d_R$.

We consider that the generalization of Corollary 2 to higher dimensions (in particular to the 3D space) is of interest, considering not only possible applications but also the theoretical significance of such an achievement. However, we note that, even in the 3D case this procedure may have rather less efficiency gains because of the lack of a natural order relation among the points lying in the central slab, and the consequent increase in the problem’s complexity.

References

Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4): 214–229, 1980. ISSN 0001-0782.

M. Smid. Closest-point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science, Amsterdam, 2000.

Michael Ian Shamos. Geometric complexity. In *STOC ’75: Proceedings of seventh annual ACM symposium on Theory of Computing*, pages 224–233, New York, NY, USA, 1975. ACM.

Jon Louis Bentley and Michael Ian Shamos. Divide-and-conquer in multidimensional space. In *STOC ’76: Proceedings of the eighth annual ACM symposium on Theory of Computing*, pages 220–230, New York, NY, USA, 1976. ACM.

Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321295358.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, USA, 2nd edition, 2001.

Qi Ge, Haitao Wang, and Hong Zhu. An improved algorithm for finding the closest pair of points. *Journal of Computer Science and Technology*, 21(1):27–31, 2006.

Minghui Jiang and Joel Gillespie. Engineering the divide-and-conquer closest pair algorithm. *Journal of Computer Science and Technology*, 22(4):532–540, 2007. ISSN 1000-9000.