Implementing Closest Pair of Points in Computational Geometry Using Divide and Conquer Algorithm

Tushar Chaudhari - 222IT007
Information Technology
National Institute of Technology,
Karnataka, Surathkal, India 575025
tushar.222it007@nitk.edu.in

Amrit Mohapatra - 222IT004
Information Technology
National Institute of Technology,
Karnataka, Surathkal, India 575025
amritmohapatra.222it004@nitk.edu.in

Neeraj Kumawat - 222IT024
Information Technology
National Institute of Technology,
Karnataka, Surathkal, India 575025
neerajkumawat.222it024@nitk.edu.in

Prof. Ram Mohana Reddy Guddeti
Information Technology
National Institute of Technology,
Karnataka, Surathkal, India 575025
profgrmreddy@nitk.edu.in

***Abstract*—In 1975, Ian Shamos published a computational geometry classic work in which he described an algorithm which has optimal time complexity O(n log n) which solves the Closest-Pair problem 2D plane. The Voronoi polygons served as the foundation for this method. Jon Bentley and Ian Shamos proposed the first optimal solution which solves the Closest-Pair problem in any dimension k $\geq$ 2. By examining a sparsity constraint generated over the set of points in the k-plane,Using a divide-and-conquer technique, those authors were able to extend the planar method to higher dimensions. We used a divide and conquer method up to three dimensions, then KDTree to reduce time complexity for larger datasets.**

Keywords: Closest Pair, Computational Geometry, Algorithms, Divide and Conquer, N-Dimensional Plane.

I. Introduction

To provide just one example of a fundamental problem in computing field in Geometry, the nearest Points Pair entails locating the two points that are the closest to each other. If we have set of points say S where number of points n 2 points in Rd , your goal is to identify the nearest pair of S , that is a points pair q0 and q1 such that p0 and p1 are distinct and not the same , the distance between p0 , p1 is should not be more than any distinct pair of points in S. Some famous Problems in Computational geometry are to find Shorter Paths in the plane which contains polygonal obstacles, Closest pair of points, Convex Hull problem ... This paper demonstrates to find minimum distance between two points which are closer in the N-dimensional plane, as well as the Divide and conquer strategy and KDTree for higher dimensions.

A. Need of CP Algorithm

The following are some of the qualities that necessitate the evaluation, refining, and development of novel closest pair algorithm:

- Closest pair problem is widely used in two most important areas- computational geometry and operational research.
- Writing cost-efficient algorithms is one of the keys to succeed as a data scientist in computational geometry as it helps in solving real life problems.
- When determining the points that are closest to one another on a plane, the more points in the dataset, the longer it takes for algorithms to find the pair having smaller distance between them.
- In everyday life, we calculate the shortest distance between two things to save time and effort.

Three dimensional closets pair of points can be analyzed using growing ball problem. In a three-dimensional space (x, y, z), there are lots of identical balls which are hanging in free space. Supposing that balls start to enlarging at the same time and with the same speed.we have to determine the initial distance between first pair of balls which are going to touch to each-other by analyzing the data.
This developing ball issue is essentially a three-dimensional closest pair problem. Due to the constant expanding speed, we must determine which ball pairings are nearest to each other in this issue in order to calculate the distance between first contacting ball pairs. We can use the naive method to pick all balls one by one and find the distance between each picked ball and the remaining balls. However, the temporal complexity of this technique is $O(n^2)$, implying that it is inefficient with large numbers which results in increasing time complexity. To tackle this difficulty, we can instead use a divide and conquer algorithm design. We can solve it in O(nlogn) time complexity with this strategy. It is possible to accomplish this by recursively partitioning our data into left and right sets.

Once the objects in the set are small enough, we may apply the naive technique, also known as the bruteforce method, to discover the shortest distance between them.

II. Literature Survey

In their study, Zhou, Xiong, and Zhu [4] emphasised how important it is to look at the all Euclidean distances which is used to find the pair of closest points. This is because such calculations are generally more expensive than basic operations like addition, subtraction, and comparison.

Bentley and Shamos [5] showed up with the first edition of the solving the 2D closest pair problem algorithm. The algorithm is made up of a step for preprocessing and a step for recursion. Inside the merge step of the existing algorithm, the distances between each point p in Y and the next seven points are calculated . this algorithm will be called Basic-7.

Instead of Y_l and Y_r., Jiang and Gillespie's proposed heuristic so that to find smaller distance in the recursion process which is same as Euclidean distance. This makes the two arrays Y_l and Y_l smaller, which reduces time in combination step. This method is Adaptive-2 when the heuristic is used in conjunction with Basic-2.

Zhou and Yu presented Pair-pruned CP, a method that works on the brute-force technique also minimises the calculations by utilising the least distance calculated. According to empirical investigation, the average calculations are having time complexity in order of O(1), implying that the temporal complexity of this method is $O(n^2)$.

CP algorithm with pair pruning. Zhou and Yu wrote about an algorithm called Pair-pruned CP. It is a brute-force approach that decreases computations by considering the shortest distance got so far. Pair-pruned CP is solved in $O(n^2)$ time.. The average number of distance calculations was found to be on the order of O(1).

we will use the Basic 7 which is classic algorithm that uses divide-and-conquer strategy, it is enough to figure out the distance from point q to no more than three points after q in direction of Y-axis. we can say this Basic-7 algorithm as Basic-3.

III. Problem Statement

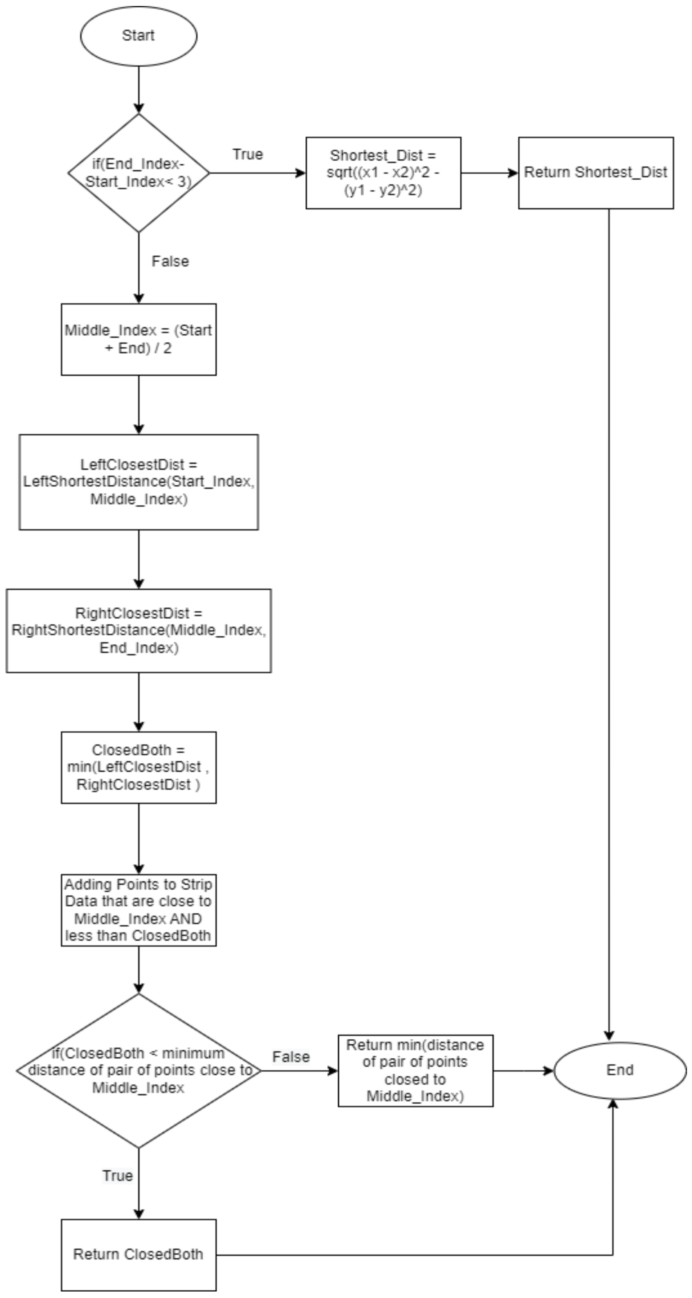
To design and develop closest pair of points algorithm using divide and conquer strategy for large number of points by finding median using divide and conquer strategy.

A. Objectives

- Design of Closest pair of points using divide and conquer algorithm for n points in 2 Dimensions.
- Optimizing the algorithm to get pair of points which are closer to each other in N dimensions.
- Using the above algorithm to implement Design of Closest pair of points using divide and conquer algorithm for n points in 2 Dimensions.
- To create a graphical user interface for calculating the shortest distance between two points out of a total of 10,000 points.

IV. Methodology

Coming to growing ball problem the tricking part is a pair of balls with minimum distance may have a ball which is on a side, and the other ball which is on the other side. We have to consider that tricking part in merging process to get the right result.



Start
if(End_Index-Start_Index< 3)
True
Shortest_Dist = sqrt((x1 - x2)^2 - (y1 - y2)^2)
Return Shortest_Dist
False
Middle_Index = (Start + End) / 2
LeftClosestDist = LeftShortestDistance(Start_Index, Middle_Index)
RightClosestDist = RightShortestDistance(Middle_Index, End_Index)
ClosedBoth = min(LeftClosestDist , RightClosestDist)
Adding Points to Strip Data that are close to Middle_Index AND less than ClosedBoth
if(ClosedBoth < minimum distance of pair of points close to Middle_Index
False
Return min(distance of pair of points closed to Middle_Index)
End
True
Return ClosedBoth

Fig. 1. Flow Chart

To solve this problem, we should create a new set called strip including elements which are close enough to middle line. Luckily, we do not have to use naive method for all elements inside strip. If we observe the elements inside strip, we will notice that there is a sparsity. The elements inside same left or right side of strip cannot have a distance less than

TABLE I
SUMMARY OF LITERATURE SURVEY

Authors	Paper Title	Conference/Journal Name	Summary
Zhou, Y., Yu, H	An efficient comparison-based deterministic algorithm to solve the closest pair problem	8th International Conference on Intelligent Computation Technology and Automation,2015	The proposed deterministic Pair-PrunedCP algorithm for solving the closest pair problem is correct, simple and easy to understand.
X. Cai, S. Rajasekaran, and F. Zhang	On the Efficient approximate algorithms for the closest pair problem in high dimensional spaces	Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 2018	This paper address the problem when the metric space is of a high dimension. To solve this problem under the 2 norm, we present two novel approximate algorithms.
S. Rajasekaran, S. Saha, and X. Cai	Novel exact and approximate algorithms for the closest pair problem	2017 IEEE International Conference on Data Mining (ICDM). IEEE	An elegant exact algorithm called MPR for the CPP that performs better than MK approximation algorithms for the CPP that are faster than MK by up to a factor of more than 40, while maintaining a very good accuracy..
Daescu, O., Teo, K.Y	2D closest pair problem: a closer look	Proceedings of the 29th Annual Canadian Conference on Computational Geometry,2017	Experimental results comparing the resulting improved version of the divide-and-conquer algorithm to other known variants in the literature for the planar closest-pair problem.

half of strip width (strip distance) with same side elements. Because, if such distance would occur, the strip width would change as well. In my solution, after the program generates left and right sets, it divides Z axis into pieces which I call columns at strip part. They have xyz dimensions of 2d, inf, 1d and numbered from increasing z order. Then program puts each element inside strip into the its proper column. My first key observation is if an element is inside a column numbered i, the other element with minimum distance should be inside in i-1, i or i+1 numbered columns (Left column of the main column, main column itself and right column of main column). Therefore, program only needs to check 3 columns at most for each ball in strip. The strip set which the program creates is already sorted in increasing Y order. In first phase of strip process, we put values from strip to correct columns. This process makes elements inside columns sorted in Y order too without calling another sorting process. The program also saves its position in the middle(main) column as well as position of last element inside left and right columns. That's because, the program needs to remember its position in increasing Y order inside column for all three columns. In seconds phase of strip process, program starts to calculate distances between elements. It starts with picking the first element of strip again. It checks the elements at upper position of this element by checking columns. If the Y distance is in range, then it calculates distance. If it is not in Y range, it means program does not need to check remaining values in the column anymore and simply breaks the loop then check another column. That means we only check a fixed volume for each ball, and since balls can be inside that area is limited, real complexity of that process is actually constant. The program does not need to check elements under (related to Y axis) the current element, since we are investigating from minimum Y valued element to maximum Y valued one progressively. Strip process returns if there is such minimum distance inside strip, otherwise return the already known minimum value which is half of strip width.

k-d tree is an established data structure which enables the use of a plethora of efficient algorithms of huge practical interest, for instance search algorithms and nearest neighbor queries [1]. Another important aspect is the existence of efficient utility algorithms which make quite efficient the update of the data structure in response to changes in the dataset (deletion, insertion, ...). In this brief report we present and analyze a parallel implementations of k-d tree which uses two standard frameworks for parallel programming, namely OpenMP (shared memory) and MPI (distributed memory). Our implementation uses the same approach for parallelization in both cases, even though some compiler-level flags can be used to activate different code regions whose goal is to fix performance bottlenecks peculiar of a framework (for instance false-sharing in OpenMP). We talk briefly about this point below.

First things first: we briefly discuss the data structure and the invariants to be maintained. Afterwards we present the very simple parallel algorithm we developed, and spend a few words on the implementation. We then procede with some

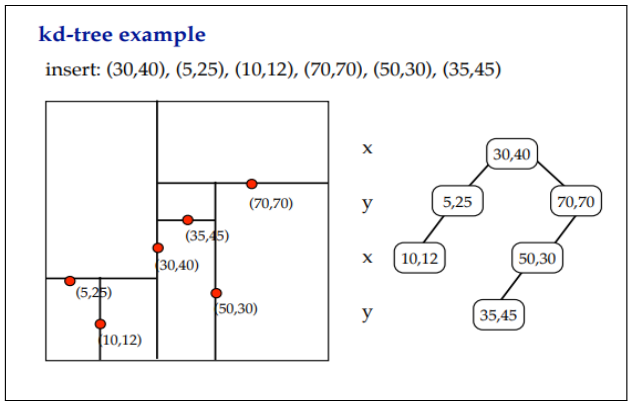


Fig. 2. KD tree insertion visualisation

considerations on our expectations for the performance, and verify our hypothesis against the real data we extracted from a set of run of the code.

Given a set of k-dimensional points $P = \{p^1, \ldots, p^n\}$ such that $p^i \in \mathbb{R}^k$ we pick a recursive definition of a k-d tree 4 . The node of the tree associated with the set P is given by:

$$\mathrm{Node}(P) = \begin{cases} \text{null} & \text{if } P = \emptyset \\ \{p_m, \mathrm{Node}\,(P_1)\,, \mathrm{Node}\,(P_2)\} & \text{otherwise} \end{cases}$$

where p_m is the median point of P against some axis i which is chosen via an unspecified criteria that we are going to formalize in some lines. $P_1, P_2 \subset P$ are defined as follows:

$$P_1 = \{p \in P \mid p < p_m\} \quad P_2 = \{p \in P \mid p > p_m\}$$

i.e. they are the subsets of points of P which ”fall” respectively before and after p_m along the i-th axis. For simplicity we assumed that P does not contain repeated values, however the definition is easily generalized. $\mathrm{Node}\,(P_1)\,, \mathrm{Node}\,(P_2)$ in (1) are respectively the left and right branch which originate from $\mathrm{Node}(P)$.

The axis i is chosen in such a way that the spread of the points in P is maximum along i. However, since we assumed that our points are distributed somewhat uniformly in the space $\mathbb{R}^k$, we are going to use a very simple function to determine the axis i used to ”split” the tree:

$$i(\ell) = \ell(\mathrm{mod} k)$$

where ℓ is the current level of the tree (i.e. the distance of the current node from the root).Figure 1 shows KDTree example.

Inserting Node in KDtree:

```
insertKDTree(Point pt, KDNode r, int cd)
  **if r is null**
    r = new KDNode(pt)
  **else if (pt is r:data)**
    // error! duplicate
  **else if (pt[cd] $\leq$ r:data[cd])**
    r.left = insertKDTree(pt, r:left, (cd+1))
  **else**
    r:right = insertKDTree(pt, r:right, (cd+1)
```

Using a visualizer tool written in Python which we developed for the occasion, we show the progression of the construction of a k-d tree for a very simple dataset. The visualization in Figure 1 renders in a very clear way the fact that points added to the tree in the level k need to satisfy constraints given by all the median points points in the parent branches in levels $0, \ldots, k-1$, against all the axes considered up to this level. Each surface is the plane containing one of the points added to the tree at some point of the algorithm, and divides the volume dedicated to the corresponding branch in two halves which shall contain all the other points (respectively) in the left and right branch.
The steps in divide and conquer of closests pair of algorithm along with pictorial representation is as follows:

The shortest distance between any two locations in the

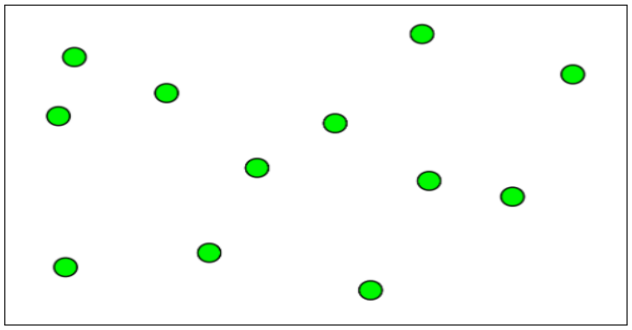


Fig. 3. Points in 2D plane

supplied array will be output. The input array is sorted according to x coordinates as a preliminary processing step.
step 1) first we find middle element in sorted array. we can consider p[n/2] to calculate middle point.
step 2) then the next step is dividing the array into two half.The first Splitted array ranges from the supplied array in from P[0] to P[n/2] and next sub array ranges from P[n/2+1] to P[n-1].

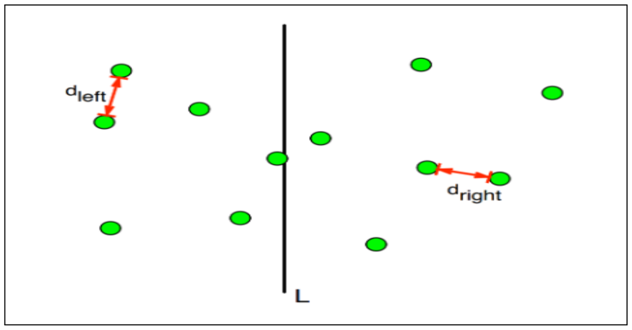


Fig. 4. Calculating distance in left and right part

step 3) This is a step that is repeated. In this case, we use recursive calls to the same function to find the shortest distances in both subarrays. We can call the distance of the left part d_l, and the distance of the right part d_r. Then we find the smaller value of d_l and d_l.

step 4) There are now three computations. We receive an upper limit d of minimum distance from the previous three operations. Now We will check the pairs in which either one point is from one side that is left or right side.it is same as we will consider one point from left and one from other side not from the same side. There is vertical line that passed through P[n/2] which is middle point,now we find any x coordinates which are nearer to the vertical line's centre than d. we will store all these points in strip[] array.

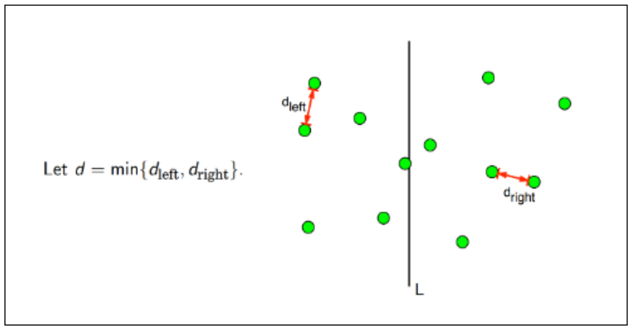


Fig. 5. Minmum distance between left and right part

step 5) In this step we need to Sort the strip[] array by y coordinates. and this sorting step takes O(nlogn) steps. here we will do recursively sort and merge , we can decrease it to O(n).
step 6) Determine the shortest distance in the strip[]. This is difficult. we can find minimum siatnce in o(n) instead of O(n2) Geometrically, There can be at most 7 co-ordinates to evaluate in strip data.
step 7) Finally, from previous step i.e step 6, we return the lowest value of d and distance.

Algorithm 1 Pseudo-code for the implementation of the Algorithm - Brute force

```
Euclidian Distance d(P$_i$, $P_j$) = $\sqrt{[(x_i - x_j)^2 + (y_i - y_j)^2]}$

**Algorithm :**
1. **Function** closestPointsBruteFoece(P)
2.    // S contains of points
3.    d$_m$ ¸$PARTITION$
4.  **for**$p = 1 to n - 1$**do**
5.        **for**$q = i + 1 to n$**do**
6.          $dis ¸ sqrt((x_p - x_q)^2 + (y_p - y_q)^2)$
7.          $if dis < d_m then$
8.              $d_m = dis;$
9.              $ind1 = p;$
10.             $ind2 = q;$
11.  $return$   $ind1, ind2$
```

The use of overwhelming force is the first strategy that springs to mind when trying to solve a problem. When the method is performed on a bigger collection of data points, the complexity of the computation takes more time, which increases the expense of using the brute force technique, which yields correct results. We are going to employ two different for loops in this strategy, which will result in a time complexity of $O(n^2)$. Every time, we will choose two points from a set of n points and then calculate the distance between those two places. Out of the total number of points, we selected $\binom{n}{2}$ pairings. The outer for loop selects one location, and the inner for loop continues processing from there.

takes every other points one by one and calculates the distance between every point. The minimum distance is then computed between all these points. The distance in x-y 2D plane is calculated using the formula that we used to calculate euclidean distance.

Algorithm 2 Pseudo-code for the implementation of the Algorithm - Divide and Conquer

```
▷$S$ is a set that consists of at least two points on the plane and is defined by the $x$ - and $y$-coordinates.
    $S_x$ := the list of points in $S$ arranged according to their $x$-coordinates
    $S_y$ := the list of points in $S$ arranged according to their $y$-coordinates
return RecClosestsPair $(S_x, S_y)$

RecClosestsPair $(P_x, P_y)$
▷$S_x$ and $S_y$ are lists of points (more than 2 )sorted by $x$ - and $y$-coordinate.
if $|S_x| \leq 3$ then
    compute pairwise distances and return the pair with closests distance
else
    $Li_x$ := first part of $S_x$; $Ri_x$ := second part of $S_x$
    $mid := (\max x\ in\ \text{Li}_x + \min x\ \text{in}\ Ri_x)/2$
    $Li_y$ := sublist of $S_y$ of points in $Li_x$;
    $Ri_y$ := sublist of $S_y$ of points in $Ri_x$
    $(m_L, n_L) :=$ RecClosestsPair $(Li_x, Li_y)$;
    $(m_R, n_R) :=$ RecClosestsPair $(Ri_x, Ri_y)$
    $\delta := (d(m_L, n_L), d(m_R, n_R))$
    if $d(m_L, n_L) = \delta$ then
        $(m^*, n^*) := (m_L, n_L)$
    else
        $(m^*, n^*) := (m_R, n_R)$
        $C$ := sub-list of $S_y$ of points whose $x$-coordinate are within $\delta$ of $mid$
        for each $b \in C$ in order of appearence on $C$ do
        each of the (up to) next seven points $c$ after $b$ on $B$ do
                if $d(b, c) < d(b^*, c^*)$ then
                    $(m^*, n^*) := (b, c)$
return $(m^*, n^*)$
```

All the steps of divide and conquer algorithm are discussed above. The divide and conquer alogrithm pseudocode is explained in above 7 steps.

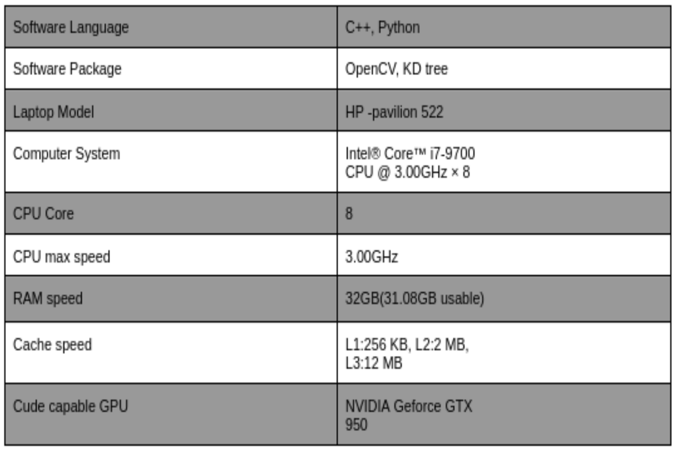
V. Datasets

We successfully tested the ball growing method on small and big datasets with points ranging from 10 to 25000. When compared to the bruteforce method, the divide and conquer technique on 3 dimension closets pair of points produces better results. We investigated similarity for huge datasets using Python random number library creation up to 100000 points. If we analyze bigger datasets and use the KDtree data structure, the nearest pair of points gives an extremely excellent result. Even when the dimensions are expanded, the results are still excellent. We tested our algorithm up to ten dimensions and hundreds thousand points.

VI. Experimental Results and Analysis

A. Computer system specifications and Software specifications

Experimental setup for the implementation i.e all the system and software specifications are mentioned in Table 2 given below.



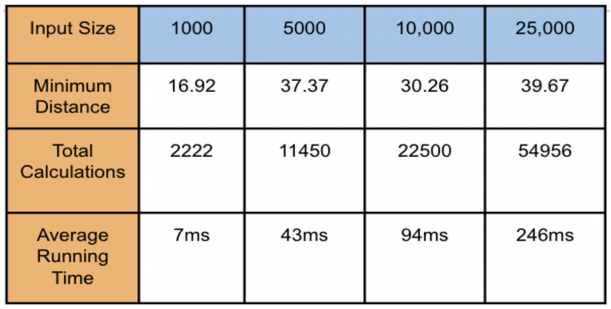
Software Language	C++, Python
Software Package	OpenCV, KD tree
Laptop Model	HP -pavilion 522
Computer System	Intel® Core™ i7-9700 CPU @ 3.00GHz × 8
CPU Core	8
CPU max speed	3.00GHz
RAM speed	32GB(31.08GB usable)
Cache speed	L1:256 KB, L2:2 MB, L3:12 MB
Cude capable GPU	NVIDIA Geforce GTX 950

Fig. 6. Software Specifications

B. Observation during experiments

First, we will examine the experimental analysis for the ball expanding algorithm, which finds the closest 3D coordinate pair. When coupled, the process for growing balls gives the initial coordinates of balls that will collide. We tested this strategy for 1000, 5000, 10,000, and 25.000 points. The algorithm returns the minimum distance between two places, the number of calculations, and the program's runtime. Dataset size increases programme completion time.

First, we will look at an experimental examination of the ball expanding method, whose goal is to locate the pair of 3D co-ordinates that is the nearest to each other. The algorithm for growing balls yields a pair that, when combined, will offer the initial coordinates of balls that will collide with one another. This approach has been evaluated for 1000, 5000, 10,000,



Input Size	1000	5000	10,000	25,000
Minimum Distance	16.92	37.37	30.26	39.67
Total Calculations	2222	11450	22500	54956
Average Running Time	7ms	43ms	94ms	246ms

Fig. 7. Perfomance comparison of serial execution with parallel(CPU)

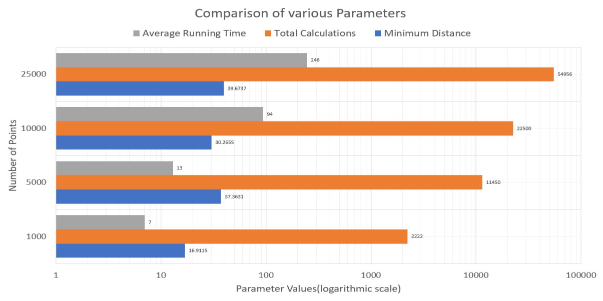


Fig. 8. Perfomance comparison of serial execution with parallel(CPU)

and 25.000 points, respectively, in our testing. The algorithm returns the smallest possible difference in distance between any two points, as well as the total number of distinct calculations and the amount of time it took to run the programme. The length of time needed to finish the programme grows proportionally with the amount of the dataset.
For 10 points with various dimensions this is the graph.

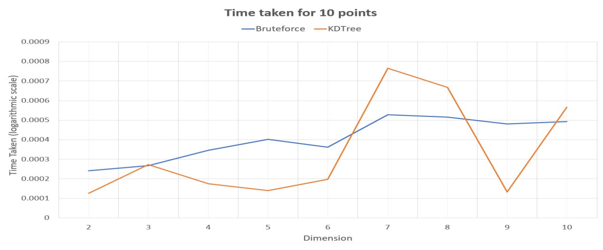


Fig. 9. Perfomance comparison of serial execution with parallel(CPU)

For 10 points with various dimensions this is the graph.

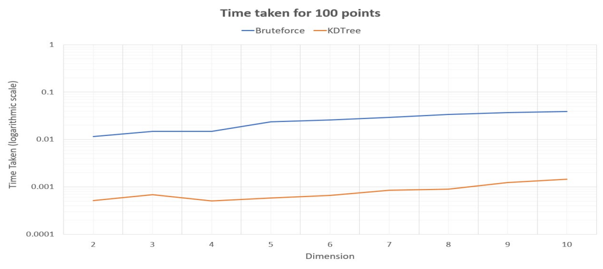


Fig. 10. Perfomance comparison of serial execution with parallel(CPU)

For 10 points with various dimensions this is the graph.

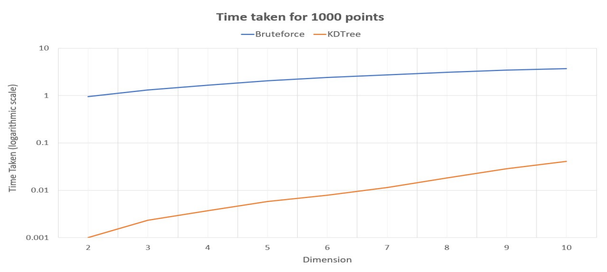


Fig. 11. Perfomance comparison of serial execution with parallel(CPU)

For 10 points with various dimensions this is the graph.

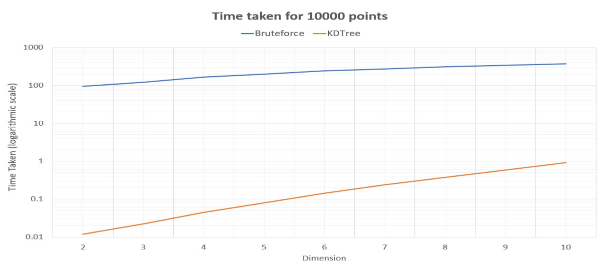


Fig. 12. Perfomance comparison of serial execution with parallel(CPU)

C. *Visualization of Algorithm*

In the next paragraphs, we will look at a visualisation of the closest pair of algorithm, which is composed of three steps. 1) First we divide all points using median of co-ordinates 2) Determine the shortest distance within each bounding box. 3) Examine the smallest distance between the left and right bounding boxes and the points on both sides of the median coordinate line.

At the very end of the process, the visualisation of the algorithm returns the final minimum distance between the two points that are the closest together.

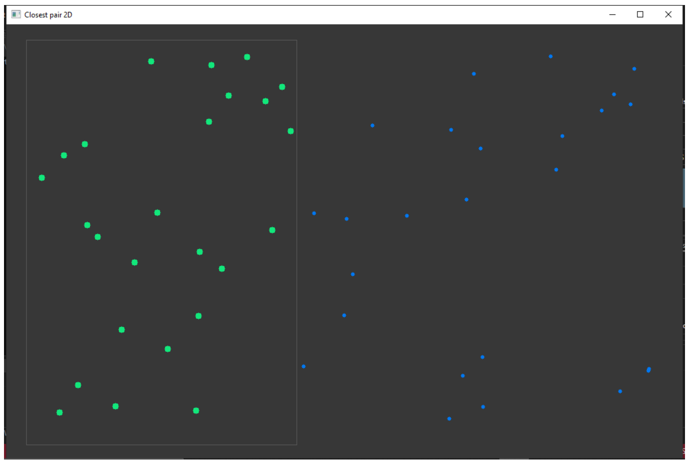


Fig. 13. Speedup Factor with respect to existing sorting techniques

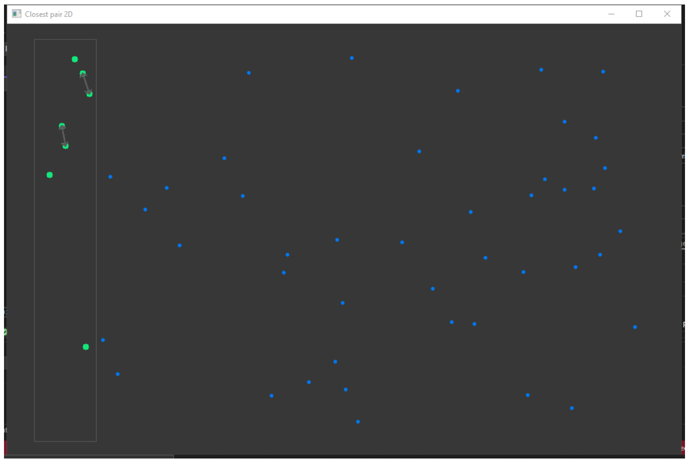


Fig. 14. Speedup Factor with respect to existing sorting techniques

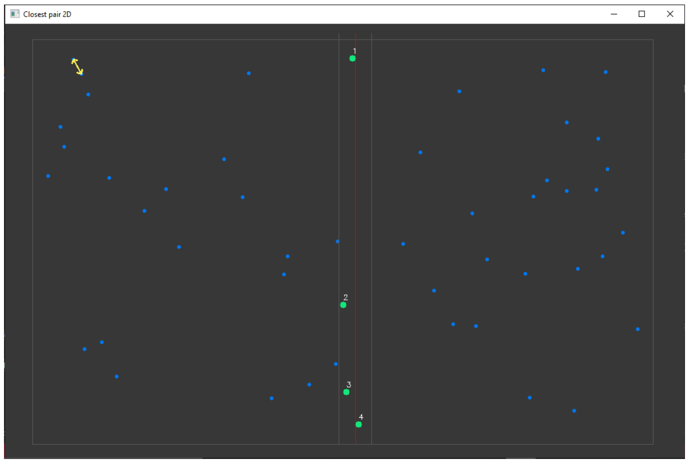


Fig. 15. Speedup Factor with respect to existing sorting techniques

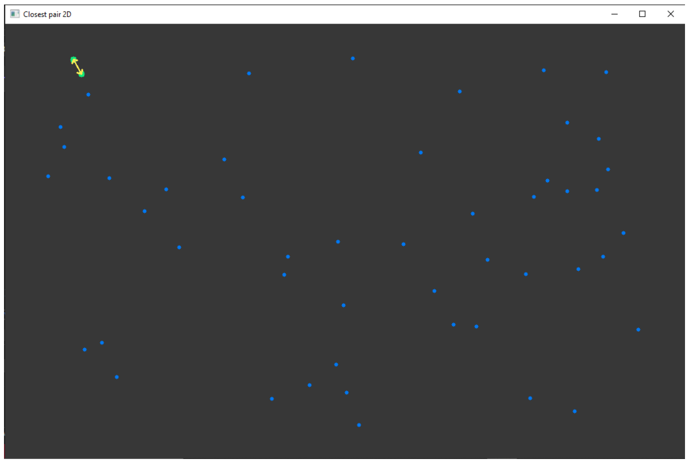
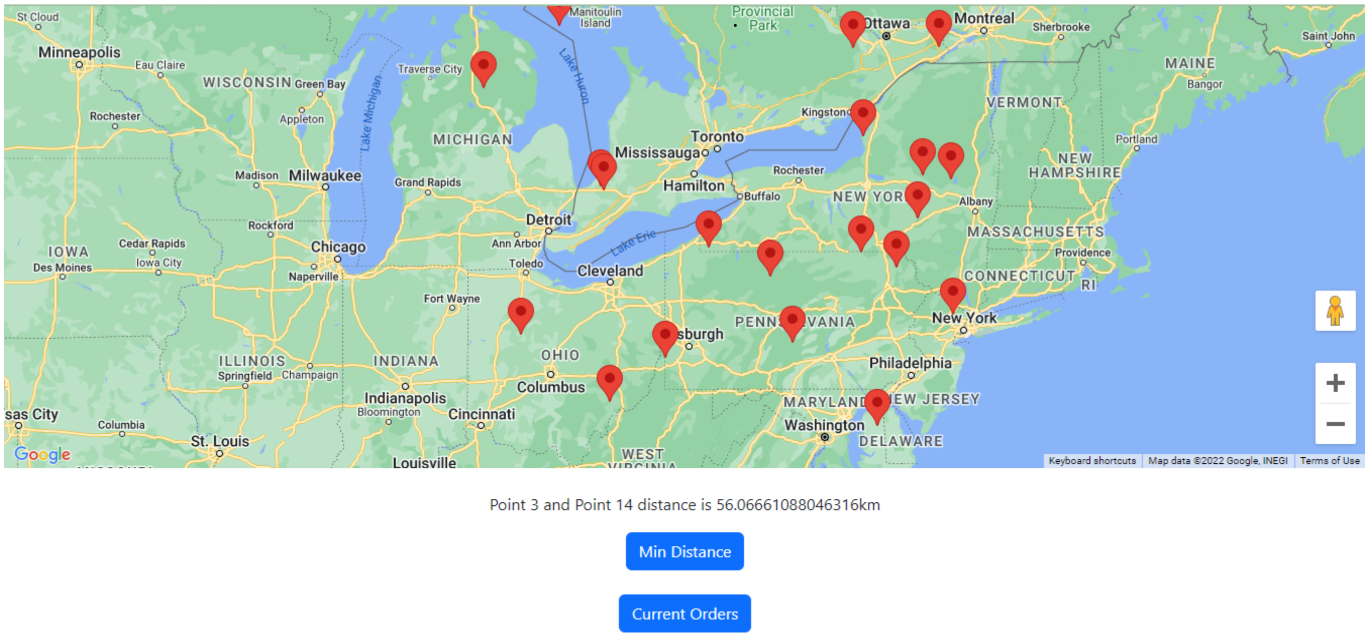


Fig. 16. Speedup Factor with respect to existing sorting techniques



Point 3 and Point 14 distance is 56.06661088046316km

D. Application

The application considered food delivery. On the front end, we display a Google map with many markers representing current restaurant orders. Clicking ”get minimum distance” displays the shortest distance between two markers in kilometres. The haversine formula can calculate the great-circle distance between two points on the sphere if their longitudes and latitudes are known. It is a specific example of a formula that is employed in spherical trigonometry, and it plays a significant role in the process of navigation. Spherical trigonometry is a subfield of spherical geometry. The distance between these map markers is determined by the algorithm in a divide-and-conquer way, which allows it to function in O(n logn) time for huge datasets. The haversine distance formula is used to determine the shortest possible distance between any pair of coordinates. We also employed the normal Euclidian distance, but the havesine formula produced more accurate and specific results.

E. Likert Analysis

Likert Scale can be described as a scale which shows what is the opinion of people with respect to some questions or statements that they are asked. Likert analysis can be of 7 points but it is usually a scale of 5 points where people are asked whether they strongly dislike or dislike or neutral or like or strongly like on the facts or questions that have been asked. In this paper we have shown on what scale people agree on our UI design, Visualization and algorithm implementation.

F. Time Complexity analysis

Brute Force Algorithm : In brute force algorithm we compute the distance between every pair of distinct points and return the indexes of points in which the distance is smallest. This takes O(N^2) time which impacts a lot when a large dataset size is used. T(n) = O(n^2)

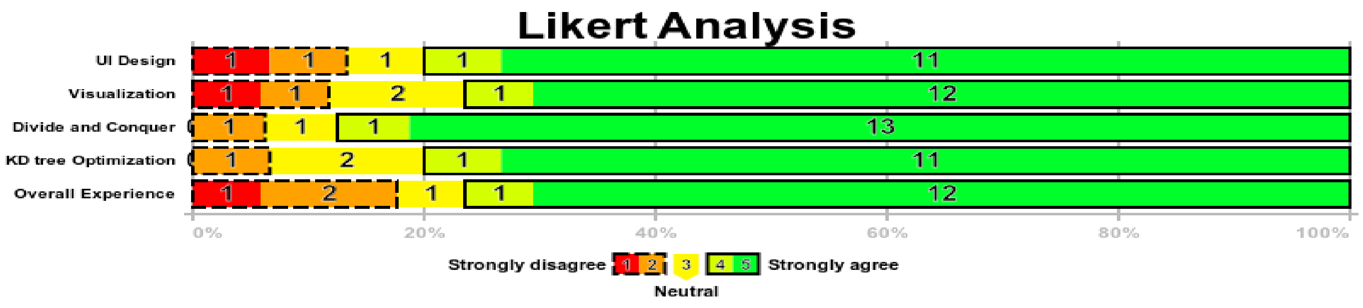


Fig. 17. Time complexity comparison with existing algorithms

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2 \sum_{i=1}^{n-1} (n-i)$$
$$= 2[(n-1) + (n-2) + \cdots + 1] = (n-1)n \in \Theta(n^2).$$

Fig. 18. Time complexity comparison with existing algorithms

Divide and Conquer: We make use of an effective divide and conquer method in order to reduce the amount of time required from O(N^2) to O(n logn). The problem is partitioned into two halves (the left half and the right half) by the algorithm, which are then solved recursively. However, this only works well when there are only two dimensions involved; as the number of dimensions grows, the algorithm becomes less effective at producing quick results.

Analysis of initial part of the algorithm :
I(n) = O(nlogn) + O(n) + O(nlogn) = O(nlogn) Recursive part of the algorithm:
R(n) = O(n) + 2R(n/2) + 4O(n) = 2R(n/2) + O(n)
If we try to solve recurrence and start iterating starting from the base value:
R(base) = 2R(base/2) + O(base)
R(base/2) = 2R(base/4) + O(base/2)
R(base/4) = 2R(base/8) + O(base/4)
It also looks like there will be logn steps. This algorithm is indeed recursive. This recurrence is in the form of the second case of master theorem with "a" is 2, "b" is 2. Therefore, we can conclude that R(n) = O(nlogn). Now we can find the final complexity of the whole algorithm:
T(n) = I(n) + R(n) = O(nlogn) + O(nlogn) = O(nlogn)
Auxiliary Space: O(log n)

Divide and Conquer using KD tree : To efficiently calculate the closest pair of points in n dimensions we we will use advanced data structure KD tree which gives O(nlogn) time for calculating closest points in any dimension. We have tested closest pair of points for higher dimensions and for more number of points using KDTree till 10 dimensions and 10000 points. T(n) = O(logn)

VII. Conclusion

The theory and implementation of the closest-pair doubling method have been briefly presented. Even though we do not have the capability of running the algorithm with a large number of points for more precise running time outputs, the method performs exactly as predicted in practise:

1. For each case with a varied number of points in P, the closest-pair distance (the same as the brute-force distance) is appropriately given without the usage of the points' coordinates. Its execution time is clearly significantly less than the result of brute-force (for large enough number of points).
2. Using KDTree, the approach can identify the closest-pair distances on Euclidean spaces of Multiple dimensions 1 without modifying any basic information. Also, even when the dimensions and number of points rise, its behaviour remains the same as when the input is a regular metric space.
3. The algorithm's running time, or execution time, is often O(n log n) in practise, as shown by theoretical aspects or the close similarity between the practical and theoretical running times regardless of the number of dimensions.

VIII. References

1. Rau, M., Nipkow, T. (2020). Verification of Closest Pair of Points Algorithms. In: Peltier, N., Sofronie-Stokkermans, V. (eds) Automated Reasoning. IJCAR 2020. Lecture Notes in Computer Science(), vol 12167. Springer, Cham. https://doi.org/10.1007/978-3-030-51054-1_20

2. Zhou, Y., Yu, H., 2015. An efficient comparison-based deterministic algorithm to solve the closest pair problem, in: 8th International Conference on Intelligent Computation Technology and Automation, pp.145–148.

3. Bertot, Y.: Formal verification of a geometry algorithm: A quest for abstract views and symmetry in Coq proofs. In: Fischer, B., Uustalu, T. (eds.) Theoretical Aspects of Computing - ICTAC 2018. LNCS, vol. 11187, pp. 3–10. Springer (2018). https://doi.org/10.1007/978-3-030-02508-3 1

4. Zhou, Y., Xiong, P., Zhu, H., 1998. An improved algorithm about the closest pair of points on plane set. Computer Research and Development 35, 957–960.

5. Bentley, J.L., Shamos, M.I., 1976. Divide-and-conquer in multidimensional space, in: Proceedings of the 8th Annual ACM Symposium on Theory of Computing, pp. 220–230.