Closest Pair

Ye Vang and Mitchell Peterson
Algorithm Design and Analysis, Shilad Sen. December 2015

Abstract

The Closest pair of Points has long been a problem of difficulty and even more so a problem of applicability.

The objective of this project was to create a solution to this problem that runs in *O(n log(n))* time rather than the more common is $O(n^2)$ time complexity.

This program is self contained, and can be especially useful for geographic matching, pairing, and other applications.

Introduction

Here's the situation: Given a list of points, find the two that are the closest. The most common solution to this is $O(n^2)$ – you need to compare every point in the list to every other one, and that takes too long as your list size goes up. We decided to try and create a faster version, with visualization and interactivity.

We chose this algorithm because it looked interesting, had real world applications, and would be challenging. For this problem, we decided that a graphical approach through Java would be our best option. We used IntelliJ as our IDE and collaborated on our code through GitHub.

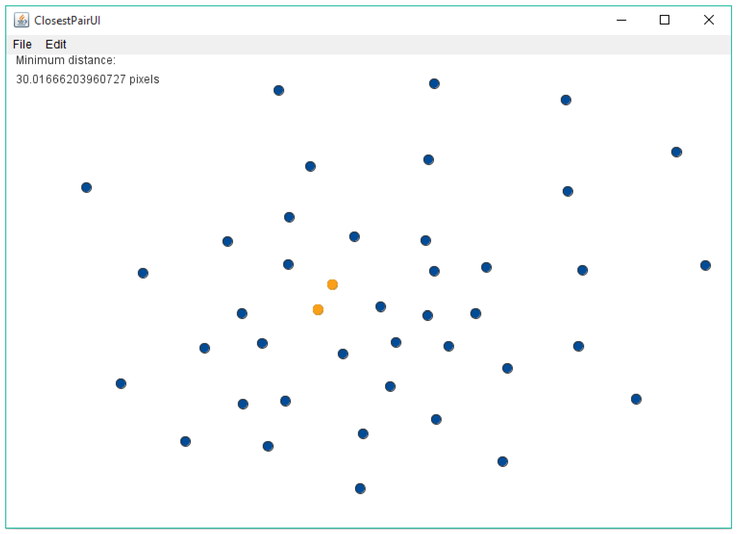

Chart 1. Screenshot of our program in action, with the closest points colored orange.

Hypothesis

To approach this problem, we originally envisioned a problem that worked like a dating website, with the algorithm finding the closest match *based around a specific point*. This isn't truly what Closest Pair is, which finds the closest pair of points in an entire data set. Closest Pair based around a specific point would be much more simple to create, since it only has to compare the given point to every other one.

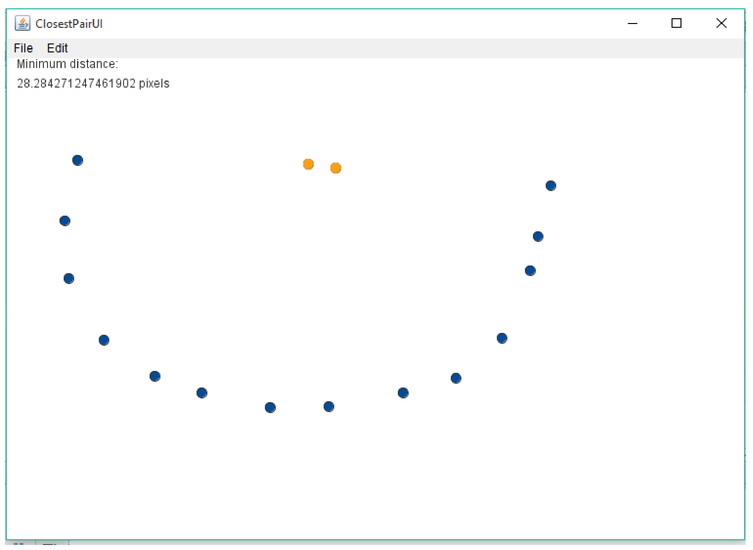

Chart 2. Second screenshot of our algorithm being run on a set of rather unique data.

Our Algorithm

We found some very general pseudocode in our book that detailed how to create an *O(n log(n)* version of this algorithm, but we tweaked it a little bit. In order to keep track of our closest pair, we made a triplet class which holds an two x-y pairs and also a distance between them. This is how we compared closest pairs and held on to the one that had the smallest distance.

The basic idea of our program was this.

1. Given a list of points, split the list into two new lists based on the X coordinates.
2. Recursively split until there are <= 3 points in the list.
3. Find the minimum distance in each set using brute force as the base case.
4. Compare each minimum distance of each set one by one.
5. Now take into consideration the Y-coordinates distances that are smaller than the shortest X distance. This will take into account any points that have a shorter distance in the y direction, or where the two points lay in between two sets.
6. Return the two closest coordinate points and the distance between them.

This runs in *O(n log(n)* because it is a divide and conquer approach, and reduces the problem size every time it runs through.

We also implemented an interactive visualization, in which the user clicks a canvas to add points. The program is constantly running in the background, and denotes which points are the closest with a different color. As the user clicks, the program updates which points are the closest, and displays the minimum distance between points at the top.

Applications

Our Closest Pair of Points algorithm has many applications, both in the real world and otherwise.

Applications we thought of include:

- Finding the closest pair of landmarks (ex. If you are on a trip to Washington D.C. and have a list of geographical locations of all the monuments, you could use this algorithm to find which two monuments to go to and still have time to check out the cherry blossoms)
- If you are using a dynamic set of points, you could use the algorithm to monitor and let you know if any two points come within a certain distance (ex. Air traffic control).
- Finding similarity of things rated by two values (ex movies, where x axis represents scariness and y axis represents comedic level).

Conclusion/Further Investigation

After designing this algorithm, we feel we have a better understanding of the Closest Pair problem, but also Divide and Conquer algorithms as a whole. We enjoyed working on the user interface, and are confident in the accuracy or our algorithm. We feel the interface is well designed, simple, and gets the job done quickly.

To take this problem to another level, we considered implementing an algorithm that finds the closest pair of points between two disjoint sets of data. This would expand its application to new heights. We could use data from geographic locations, hospitals and schools, or homes and yoga clubs, etc.

A third application we thought of was finding the closest pair in 3D space, but it ended up taking much more time than we had allotted for this project..

References

1. Levitin, Anany. *Introduction to the Design & Analysis of Algorithms*. Boston: Pearson Addison-Wesley, 2007. Print.
2. Bakhtiar, Sam. *Closest Pair: A Divide-and-Conquer Approach*. N.p., n.d. Web.