Closests Points in the Plane implementation

Arleth, Johan
Bele, Claudiu
Jacobsen, Andreas
Ulrik, Kenneth Ry

September 24, 2016

1 Results

Our implementation produces the expected results on most input files, and executes in acceptable time for all inputs except for the `wc-instance-65534` file, where our implementation appears to run for several minutes without finding a result. We believe for this to be simply too great a task to accomplish within a minute as is the target for execution of all inputs. We notice that the output-file, `closest-pair-out`, does not contain a result for this file either, however, and therefore accept that the file is perhaps not included in the outlaid one-minute execution target. An execution of all input files excluding this one file (We have removed it from the directory) – without printing enabled – finished in 29453 ms[1]. We also note that the particularly large input problem in `close-pairs-6-in` of size 20000 is computed as well, although its result does not appear in the output-file either.

We get minor decimal-differences in the files `rd100-tsp` and `u1060-tsp`. This is probably due to differences in the floating points numbers implementation between the output-file generated and the .NET platform.

Definitely wrong results are found for the files `kroA150-tsp`, `kroE100-tsp` as well as `burma14-tsp`. Also, we find the almost-correct result for file `berlin52-tsp`, where the correct result is `15` and we somehow achieve `15.8113883008419`. We have not thoroughly examined the cause for these erroneous results and thus do not know their potential cause.

2 Implementation details

2.1 Comparison upon recursive joins

First, we check for the input's size. If it contains three or less elements, we perform a n^2 check, finding the shortest distance between these one to three points. If there is only one point, the maximum value for C#'s `double` type is returned, thus ensuring a potential other result to trumpf it. The received input is assumed to be sorted by their horizontal (x) value and from there, we split it into two halves, calling recursively from there.

Upon intermediate results being returned from two halves, also called a join of the recursion results, we must consider the case where points from either half may be closer to each other than the points could possibly be individually in either half. This is the case if two points are close to

[1] All the other files beginning with "wc-" have been computed in this run.

the edge of the plane, where it was splitted; closer than any other points in the whole plane. In this case, we find the points from within the x value where the planes were splitted and fetch all points with x-values not differing more from the splitline than the best result returned from the recursive calls. These points are then sorted by their vertical (y) values, and every point is compared to the nearest eight points with smaller y-values and the nearest eight points with larger y-values. These are the points that may potentially have shorter distances in between them than what was already found to be the best recursively returned result. This reason for this specific amount of comparisons required is described in proof (5.10) in Kleinberg and Tardos, *Algorithm Design*, Addison-Wesley 2008.

Our running time is $O(n \log n)$ for n points.

2.2 Potential performance improvements

When comparing elements upon joining recursively achieved results with points between the sub-planes within delta length from the splitted edges, we currently perform an unnecessary amount of comparisons.

Theoretically, we may, for the points in this space, with potentially shorter distances inbetween, compare each one with each other only once per unique pair. This could be achieved by, when iterating the points sorted by their y-values, only compare the current point to the *next* 8 points (the maximum possible closer points, as discussed previously), and never compare to points with a higher y-value. Sadly, we failed to implement this properly in time.