**A snowball algorithm for maximal distance in a set**

By A.R.J. Marchand

CNRS UMR 5287, Bordeaux

I present here a novel snowball algorithm to compute the maximal distance between two points in a N-dimensional set. At each step, two points A and B are the diameter of a hypersphere. If the hypersphere encloses all points, the distance AB is the solution. Otherwise, the first point C not included in the hypersphere is used to search searching forward (cyclically) in the set, looking for the first point D with distance CD>AB. If no point D is found, C is eliminated and replaced by the next C. If D is found, CD is substituted to AB. The algorithm is found to run in linear time.

A straightforward search for the maximal distance in a set of size M requires M(M-1)/2 distance calculations, which correspond to class *O*(*n²*). Faster algorithms have been described, such as which runs in in dimension two.

Based on the performance of another snowball algorithm (Marchand 2023), the first objective here was to have an algorithm running in in *O*(*n*), meaning that the number of steps required to solve the problem increases linearly with the number of points. The second objective was to have a simple algorithm applicable to higher dimensions.

We start with any points A and B considered as the diameter of a hypersphere. If the hypersphere encloses all points, the distance AB is the solution, because no pair of points enclosed in the hypersphere can be further apart than the diameter AB. The first point C found not to be included in the hypersphere is used to look for the first point D such that the distance CD is strictly larger than the distance AB. If no point D is found, C is eliminated and replaced by the next point outside the hypersphere. If D is found, CD is substituted to AB and the process is iterated. The set is *always searched forward* and when it is exhausted, the search starts again at the first point.

The largest distance is unique, but the two points found as a solution may not be unique. Other points may be found by removing one or the other of these two points and starting the search again.

**Correctness**

The snowball algorithm relies on the fact that for a distance CD to be larger than AB, at least one of the points C must lie outside the hypersphere of diameter AB. Because at the next step CD>AB, the minimal distance is strictly increasing. Moreover, the algorithm only stops when no point can be found outside the hypersphere, which is why unsuitable points C must be eliminated. Together, these conditions guarantee the correctness of the algorithm, but not its performance.

**Performance**

The performance of the snowball algorithm was simulated with the Python script shown below. With n points uniformly distributed in a square or a disk (Fig. 1), execution time increased in an approximately linear way in dimensions 2 to 4 with 102 to 105 points, and slightly more than linearly in dimensions 5 to 7. This is empirical evidence that the algorithm belongs to class *O*(*n*).

The number of distances computed per point started around 6.5 in 2D and increased roughly as the square of the number of dimensions.

**Discussion**

The algorithm presents analogies with the snowball algorithm for the minimum circle, in particular because it relies on a strictly increasing radius, it uses a forward-only search, and because the search stops as soon as a point fulfilling a condition is found. This is probably the key to its efficiency.

The algorithm requires little working memory and few operations because only the current subset of two points and the center and radius of their enclosing disk need to be kept in memory at each step. It is subject to the same randomization constraints as other algorithms if the ordered set presents a spatial trend.

**References**

Marchand, A. R. J. (2023). The snowball algorithm for minimal enclosing disks.

**Annex: Python script for the snowball algorithm**