Snowball algorithms for minimal disk and maximal distance

Alain Marchand

Snowball algorithms for minimal disk and maximal distance

by A.R.J. Marchand

CNRS, UMR 5287, Bordeaux

Contact: Alain Marchand, Institut de Neurosciences Cognitives et Intégratives d'Aquitaine, Université de Bordeaux, 146, rue Léo Saignat, 33076 Bordeaux Cedex, France.

e-mail : Alain.marchand92@free.fr

http://www.incia.u-bordeaux1.fr/spip.php?article879

I introduce here two novel "snowball" algorithms. One of them very efficiently solves the problem of the minimum enclosing disk over a set of points in Euclidean space. The other algorithm addresses the question of the maximal distance between two points in a set (not necessarily Euclidean). Both algorithms tend to run in linear time in low-dimensional spaces.

This article presents two similarly simple algorithms I dubbed "snowball", to solve the minimum enclosing disk and the maximal distance between two points in a set. The snowball algorithms cyclically search through the set of elements to be explored, keeping only a small adaptive subset, while maximizing some function of distance on this subset. The search is centered on a ball in metric space and focuses on the first element that satisfies a particular distance condition. The algorithm terminates when no such element can be found. Empirical evidence indicates that these algorithms run in linear time as long as the number of dimensions is low.

I. Minimal enclosing disk

Since its first formulation by mathematician James Joseph Sylvester in 1857, the problem of the minimal enclosing disk has continued to elicit interest (Chrystal, 1885 ; Elzinga & Hearn, 1972; Megiddo, 1983; Skyum, 1991; Welzl, 1991; Efrat, Sharir & Ziv, 1994; Har-Peled & Mazumdar, 2005; Yildirim, 2008; Gao & Wang, 2018; Smolik & Skala, 2022). In two dimensions, the problem is to find a disk that encloses all the points from a set in the plane, with the smallest possible radius. Efficient algorithms are needed since the brute force solution, i.e. testing all possible circles based on two or three points, is not practical for large numbers of points.

The current standard solution is that proposed by Welzl (1991). Welzl's algorithm runs in $O(n)$, meaning that the number of steps required to solve the problem increases linearly with the number of points. Its performance is however hampered by its recursive nature, which stems from the requirement to include at each step all the points that have been examined so far. The simple algorithm presented here avoids this requirement.

The algorithm

The snowball algorithm relies on the well-known properties of the minimal circle:

- It is unique for a given set of points in the plane;
- It is determined by three points on its circumference if these three points form a triangle that is not obtuse. Otherwise, two points suffice, determining a diameter of the minimum circle.

Thus, a minimal circle enclosing any sample of four points can always be defined by some subset of two or three of them (forming a diameter or a non-obtuse triangle). This means that one and maybe two points are unnecessary.

A new sample of four points can be obtained by adding new points to the necessary subset. This is the basis of several algorithms, including that of Elzinga & Hearn (1972). Note that in general, the unnecessary points should not be completely discarded, as they may be needed at a later step.

At each step of the snowball algorithm, after a minimal circle has been determined for by two or three points, the sample is complemented to four by the first point(s) found outside the circle in the whole set. The set is always searched forward and when it is exhausted, the search starts again at the first point. Unlike in Elzinga & Hearn (1972), no point is tested and eliminated. Unlike in Welzl (1991), no attempt is made to recursively ensure that the current circle includes all previously explored points.

The algorithm stops when no point can be found outside the current circle.

Correctness

The correctness of the algorithm is guaranteed by the following properties:

1) At every step, the circle defined by the necessary subset is the minimal circle for the subset as well as the minimal circle for the sample four points. So *if* a circle obtained in this way ultimately encloses all the points in the analyzed set, then it must necessarily be the minimal enclosing circle for the whole set;
2) At every step which includes a new point outside the circle, the radius of the new enclosing circle strictly increases;
3) The algorithm only stops when no point can be found outside the current circle;
4) The set of possible circles is finite and includes the unique solution.

Performance

The efficiency of the snowball algorithm, simulated using a Python script (Annex 1) was remarkable. With N points uniformly distributed in a square or a disk (Fig. 1), execution time increased in an approximately linear way in the range $N=10^2$ to 10^6. This is empirical evidence that the algorithm belongs to class $O(n)$.

Moreover, the radius of the circle increased rapidly at each step, so that for large sets the number of radius updates (steps) was negligible compared to the number of points. Time performance was thus dominated by the search for points outside the current circle, i.e. computing the distance (squared) from the center to the next points.

The total number of distances computed was close to 2N (Figure 2). The experimental values (2.21N and 2.08N for n=5000) compare favorably with Welzl's MTFBALL algorithm, which reports an average of 4.9 and 5.6 distance computations per point.

Other trials with Gaussian radial or annular distributions and with elongated distributions yielded essentially the same results.

Discussion

While a brute force search for the minimal enclosing circle in a set of size N requires on the order of N^4 distance calculations, the best known algorithms run in linear time. Therefore, a linear time algorithm is of great interest. It is not immediately obvious that the snowball algorithm is correct or efficient. Indeed, some points temporarily left over on a given step may be necessary at a later step. This could explain why previous researchers (e.g. Elzinga & Hearn, 1972; Welzl, 1991) overlooked this relatively straightforward method. However, I proved here that all points get eventually enclosed and that the circle found is indeed the minimal one.

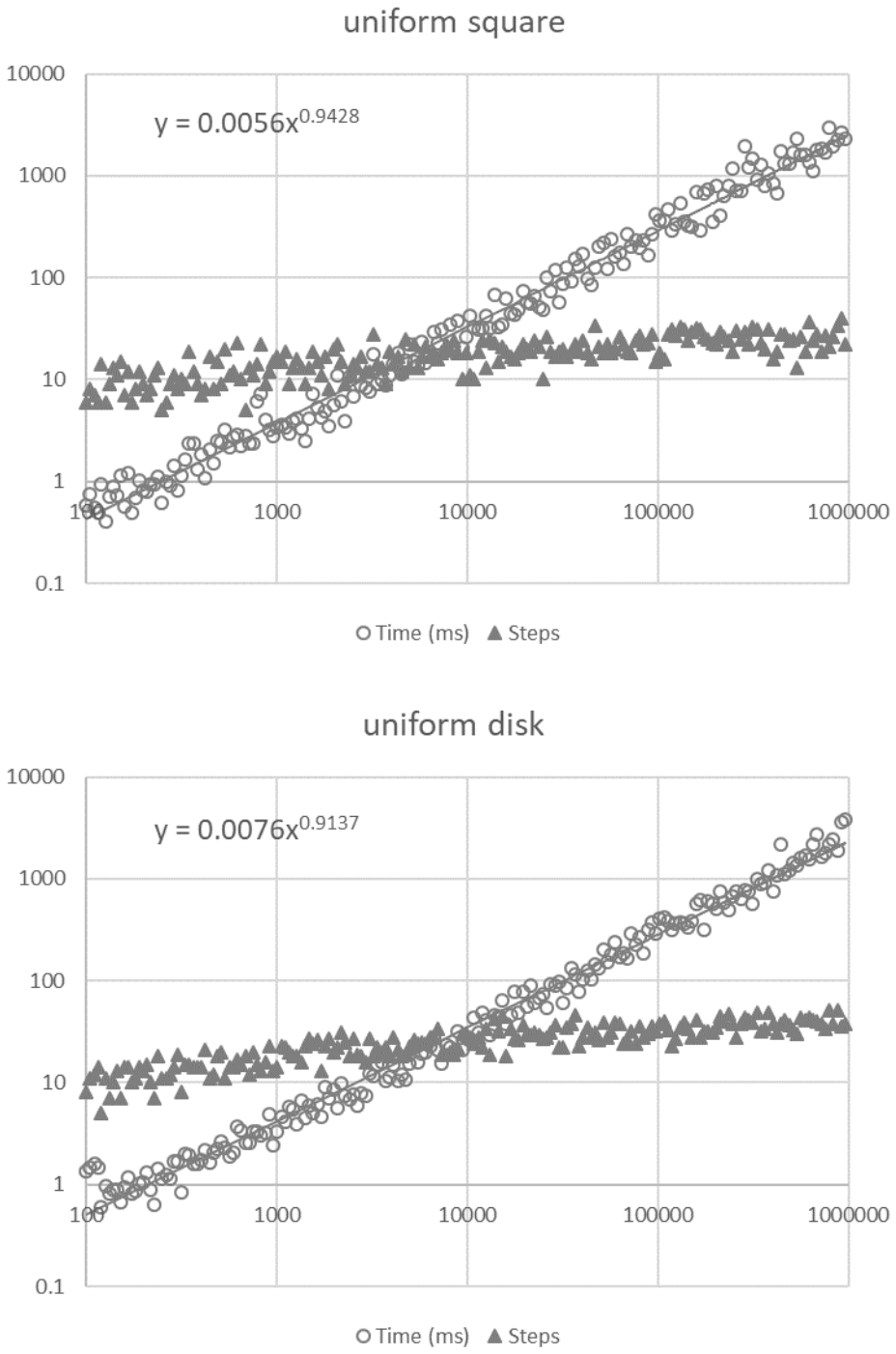

Figure 1. Time complexity and number of steps evaluated for N points uniformly distributed in a square (upper panel) or a disk (lower panel), as a function of N (log/log scale). Each symbol represents a simulation for increasingly larger sets. Execution time increased as $N^{0.94}$ and $N^{0.91}$ respectively. The number of radius updates (steps) increased much more slowly to approximately 30 and 45 respectively for $N=10^6$ points.

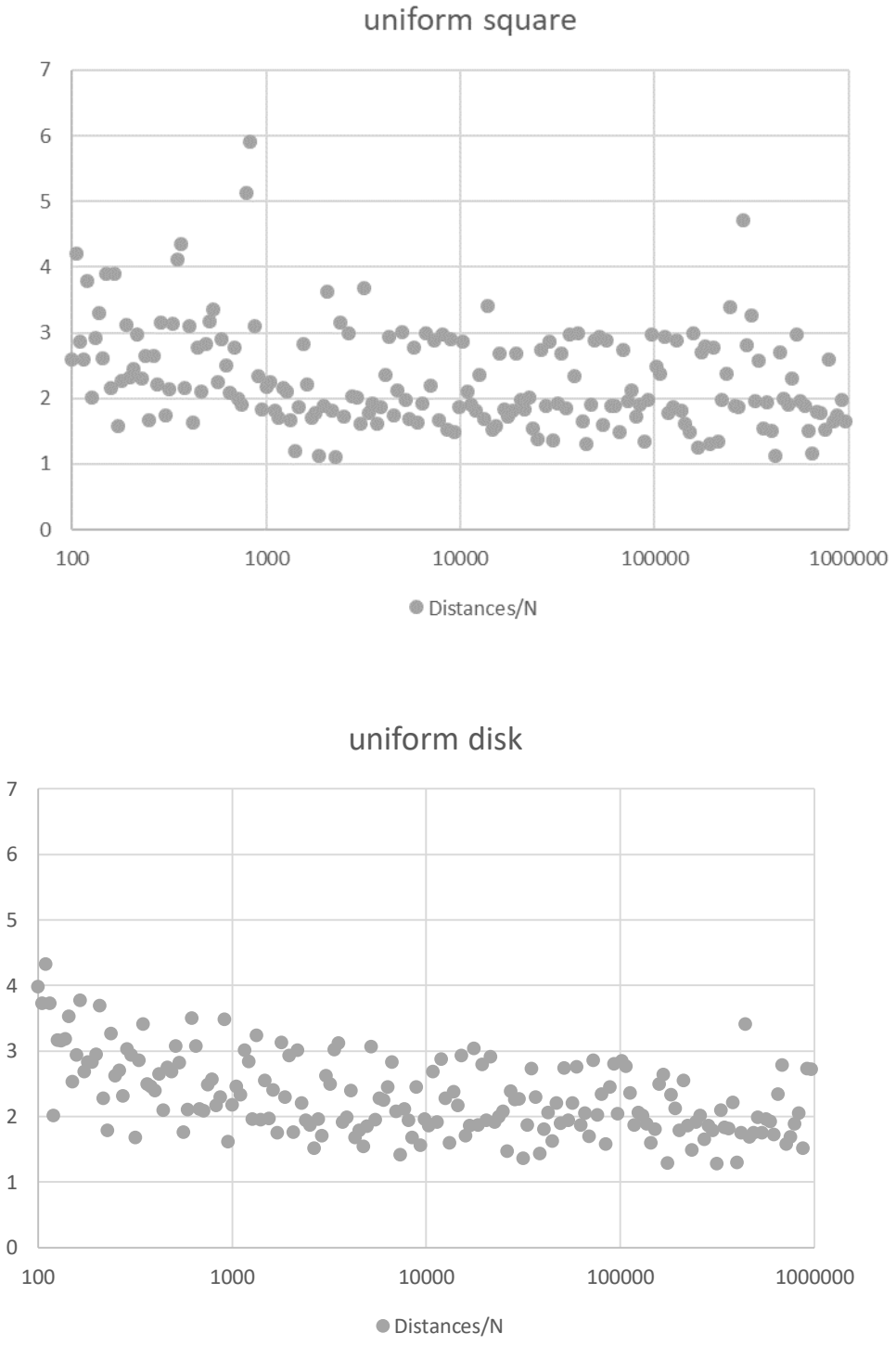

Figure 2. Ratio of the total number of distances computed over N, as a function of the size N of the set (semi-log scale). For large sets, this number is close to 2N.

Not only is the algorithm correct, but its performance is exceptional. Its time complexity appears to be $O(n)$ and it requires fewer distance computations as Welzl's algorithm.

The recursive nature of Welzl's algorithm precludes its use for very large sets because of the depth of recursion and the necessity to recursively store subsets. The snowball algorithm does not suffer from this limitation and can therefore accommodate larger sets. Points may be accessed sequentially which is another simplification.

The algorithm requires little working memory and few operations because only the current subset of two or three points and the center and radius of their enclosing disk need to be kept in memory at each step. The computation of the minimal circle of four points requires more calculations than a simple distance check, but since the number of radius updates is negligible relative to the total number of points, most of the computing time is taken by distance computations to verify the inclusion of single points.

It should be noted that attempts to speed up the search by eliminating inner points, as proposed for instance by Elzinga & Hearn (1972), Skyum (1991) or Smolik & Skala (2022), were unsuccessful. The snowball algorithm tests a distance to each point approximately twice over the whole process. This leaves little room for improvement. Indeed, at least one distance measurement per point is needed just to verify that the final circle encloses all points. So, it is not clear that a point can be eliminated with a cost of less than two distance computations. This process of removing points at each step makes Elzinga & Hearn (1972) algorithm much slower than the present one.

Of course, performance will necessarily degrade if the set of points exhibits a sequential spatial trend. This issue was already raised by Welzl who indicated that it is sufficient to randomize the set at the start. Provided that the trend is not periodic, it is even possible to avoid the cost of randomizing by browsing the set with a large step p so that p and N are relative primes and p is not too close to a divisor of N. Combined with the cyclical search, it guarantees that all points will be explored in N steps with no additional cost compared to a unit step.

Finally, the snowball algorithm, like Welzl's, should easily generalize to higher dimensions and to ellipsoids.

II. Maximal distance within a set

The problem of finding the maximal distance between two elements in a set is defined for any metric space, i.e. any set of elements equipped with a notion of distance. The most familiar example is Euclidean space, with distance based on point coordinates, but other types of distance have been considered, for instance the Manhattan (city-block) distance, or the Hamming distance that quantifies the dissimilarity between elements. Many "divide and conquer" algorithms take advantage of the topology of Euclidean space to define regions and reduce the combinatorial complexity of problems by solving them in each region. However, in more general metric spaces, notions such as straight lines or centers may have no meaning, so some of these algorithms may not be applicable.

A brute force search for the maximal distance in a set of size M is always possible. It requires M(M-1)/2 distance calculations, which place it in class $O(n^2)$. In Euclidean space of dimension two, faster algorithms have been described, such as that of Skala & Majdisova (2015), which runs in approximately linear time.

Based on the performance of the previous snowball algorithm, the first objective of the present study was to obtain an algorithm running in $O(n)$. The second objective was to have a simple algorithm applicable to higher dimensions, and possibly to non-Euclidean metric spaces.

The algorithm

The method is again quite simple. The search starts from any pair of points A and B considered as the diameter of a ball (hypersphere). If this ball encloses all points, then the distance AB is the solution, because no pair of points enclosed in the ball can be further apart than the diameter AB.

This entails that for any pair of points CD to be further apart than the diameter AB, at least one of the points C or D must lie outside the ball of diameter AB.

The first point C found not to be included in the ball is therefore used to look for another point D such that the distance CD be strictly larger than the distance AB. If no point D is found, C is eliminated and replaced by the next point outside the ball. As soon as a point 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 cyclically starts again at the first point.

The algorithm stops when no point can be found outside the current ball.

Correctness

The correctness of the algorithm is proved by the following properties:

1) for a distance CD to be larger than AB, at least one of the points C must lie outside the ball of diameter AB;
2) because CD>AB, the radius of the ball strictly increases at the next step;
3) the algorithm only stops when no point can be found outside the ball (which is why unsuitable points C must be eliminated);
4) The set of possible diameters is finite and includes the solution.

Together, these conditions guarantee the correctness of the algorithm, but not its performance.

Note that 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 repeating the search.

Performance

The performance of the snowball algorithm was simulated with a Python script (Annex 2). With N points uniformly distributed in a Euclidean hypersquare (Fig. 3), execution time increased in an approximately linear manner in dimensions 2 to 4 with 10^2 to 10^5 points. This is empirical evidence that the algorithm belongs to class $O(n)$ in low dimensional space.

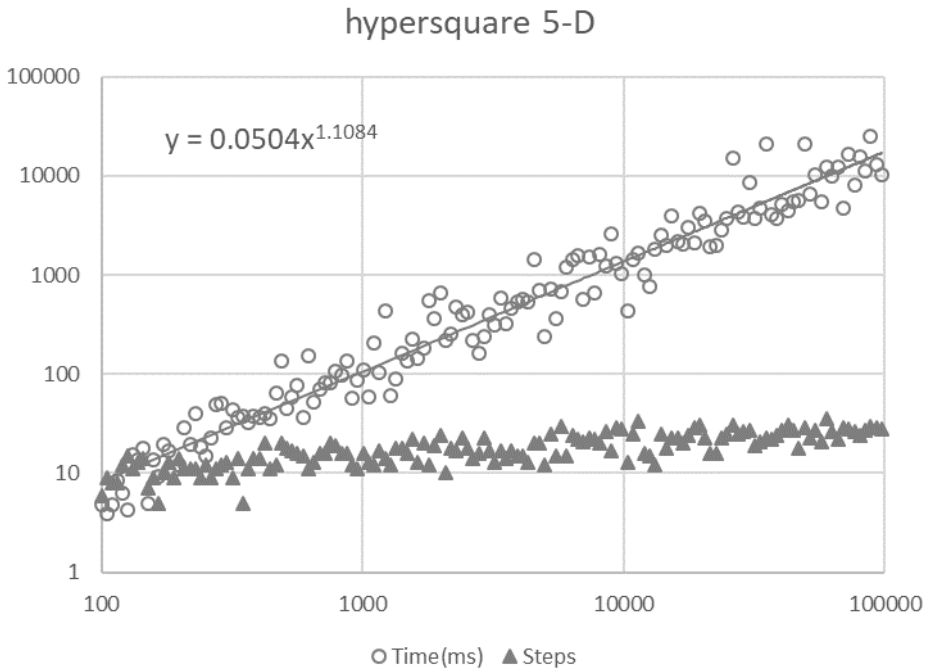

Figure 3. Time complexity and number of steps evaluated for N points uniformly distributed in a Euclidean hypersquare in 5 dimensions, as a function of N (log/log scale). Each symbol represents a simulation for increasingly larger sets. Execution time increased as $N^{1.1084}$ which is slightly above linear in 5-D. The number of steps increased slowly to approximately 27 for $N=10^5$ points.

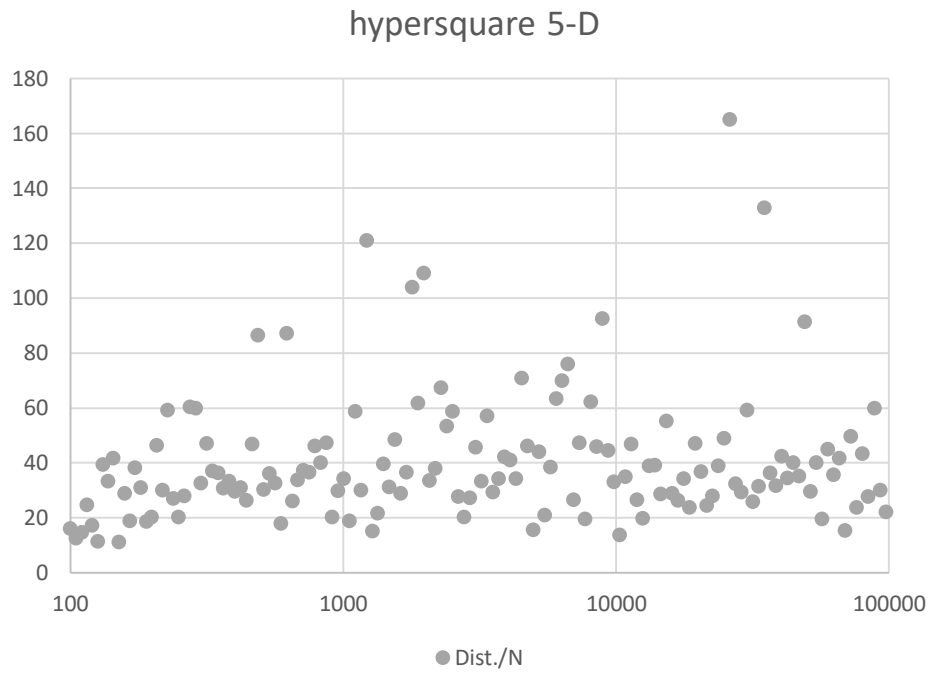

Figure 4. Ratio of the total number of distances computed over N, as a function of the size N of the set (semi-log scale). Points uniformly drawn in a Euclidean hypersquare in 5 dimensions.

The number of radius updates (steps) was negligible compared to the number of points in the set and to the number of distances that needed to be computed. The number of distances computed per point started around 6.5 in 2D and increased with the number of dimensions.

Performance in higher dimensions

Because of its simplicity, the maximum distance algorithm is well suited to examine the effect of increasing the number of dimensions. Our measure of interest was the number of distances computed, because it is the costliest and it provides an easy comparison with the brute force method.

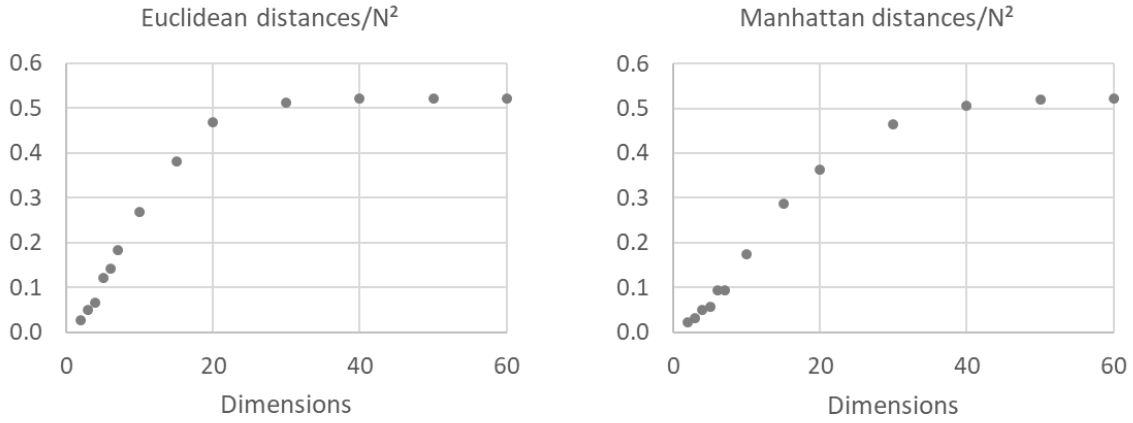

Figure 5. Ratio of the total number of distances computed over the square of the number of points N^2, as a function of the number of dimensions (N in range 100 to 1000). For higher dimensions, the number of distance calculations increased quadratically and was close to $N^2/2$, equivalent to the brute force method.

While performance was essentially linear in low dimension, it gradually became quadratic as the number of spatial dimensions increased. Above 20 (Euclidean) or 30 (Manhattan) dimensions, the number of distance calculations was slightly more than $N^2/2$, i.e. the number required by the brute force method.

By contrast, the number of radius updates (steps) was essentially constant for a given N, irrespective of the number of dimensions. This indicates that at each radius, finding a point outside the current ball becomes increasingly costly in high dimension.

Discussion

The maximum distance algorithm is very similar to the minimum circle algorithm, 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. It is subject to the same randomization constraints as other algorithms if the ordered set presents a spatial trend.

In low dimension, this snowball algorithm largely outperforms the brute force one in terms of the number of distances computed. It 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.

The maximum distance algorithm may be applied to any metric space that allows the definition of a center point for a ball. So, it cannot be directly applied to Hamming distance. On an extended version of Hamming distance, allowing virtual points midway between actual points, the algorithm performed poorly, requiring approximately N(N-1)/2 distance calculations (similar to the brute force method). This is attributable to the very restricted set of possible distance values.

Both snowball algorithms rely on finding at each step a point outside a current ball. In Manhattan and Euclidean distance, finding a point outside a current ball becomes more and more costly as the number of spatial dimensions increases. This may be explained by the central limit theorem. The Manhattan distance of random points is the sum of random variables corresponding to differences on each coordinate. Similarly, the square of the Euclidean distance of random points is the sum of random variables corresponding to the square of differences on each coordinate. So, in both cases, as the number of dimensions increases, the distribution of distances should increasingly resemble a narrow Gaussian distribution. This should lead to an increase in the number of random points that can be found within a ball defined by any pair of points, and should hinder the search for a point outside this ball.

Indeed, the maximal distance algorithm performed no better in high dimension than the brute force algorithm. This suggests that in spaces of high dimension it will be difficult to find efficient algorithms relying mainly on distance.

Overall, the simplicity of snowball algorithms may render them applicable to a variety of metric spaces, not necessarily Euclidean.

Acknowledgements

The author is grateful to Michel Carré for useful comments on the algorithm.

References

Chrystal, M. (trad. M. l'abbé Pautonnier) (1885). Sur le problème de la construction du cercle minimum renfermant n points de données d'un plan, *Bulletin de la S.M.F., Société mathématique de France* : 198-200.

Efrat, A., Sharir, M., Ziv, A. (1994). Computing the smallest k-enclosing circle and related problems. *Computational Geometry*, 4(3), 119–136.

Elzinga, J., Hearn, D. W. (1972). The minimum covering sphere problem. *Management Science*, 19: 96–104.

Gao, S.,Wang, C. (2018). A new algorithm for the smallest enclosing circle. In: 2018 8th *International Conference on Management, Education and Information* (MEICI 2018), Atlantis Press.

Har-Peled, S., Mazumdar, S. (2005). Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3), 147–157.

Megiddo, N. (1983). Linear-time algorithms for linear programming in R3 and related problems. *SIAM Journal on Computing*, 12(4), 759–776.

Skyum, S. (1991). A simple algorithm for computing the smallest enclosing circle. *Information Processing Letters*, 37(3), 121–125.

Skala, V., Majdisova, Z. (2015). Fast Algorithm for Finding Maximum Distance with Space Subdivision in E2. *In: Zhang, YJ. (eds) Image and Graphics.* ICIG 2015. Lecture Notes in Computer Science, vol 9218. Springer, Cham.

Smolik, M., Skala, V. (2022). Efficient Speed-Up of the Smallest Enclosing Circle Algorithm. *Informatica*, 33(3): 623-633.

Welzl, E. (1991). Smallest enclosing disks (balls and ellipsoids). *In: Maurer, H. (Ed.), New Results and New Trends in Computer Science*, Lecture Notes in Computer Science, Vol. 555. Springer, Berlin, Heidelberg.

Yildirim, E. A. (2008). Two algorithms for the minimum enclosing ball problem*, SIAM J. Optim.*, 19: 1368–1391.

Annex 1: Python procedure for the minimum disk algorithm (2-D Euclidean)

The complete program is available on: https://github.com/MarchandAlain/Snowball

```
def minimum_disk (points, subset, position):
  """
  Parameters: points, a list of points, tuples of 2 float
            subset, a list of four points to start with
            position, in points list
  Returns: subset, a list of two or three points
  O, a point, tuple of 2 float, center of the circumscribed circle
            r_squared, a float
  """
  while True:

    # try solving with 2 points
    P, s, [A, B] = min_diameter(subset)
    if P:
      O, r_squared = P, s
      subset = [A, B]

    # solve with 3 points
    else:
      O, r_squared, [A, B, C] = min_circumscribed(subset)
      subset = [A, B, C]

    # try adding one outside point
    D, position = point_outside(points, position, O, r_squared)
```

```
    if not D:
      return subset, O, r_squared                    # all points are inside: return
    else:
      subset += [D]

    # try completing to four with another outside point
    if len(subset) == 3:
      D, position = point_outside(points, position, O, r_squared)
      if D:
        subset += [D]
```

Annex 2: Python procedure for the maximum distance algorithm (2-D Euclidean)

The complete program is available on: https://github.com/MarchandAlain/Snowball

```
def maximum_distance (points, diameter):
  """
  Parameters: points, a list of points, tuples of 2 float
              diameter, a list of two points to start with
  Returns: diameter, a list of two points
              r_squared, a float, one fourth the square of diameter
  """
 while True:
      O,r_squared = circle(diameter)

      # try finding one outside point
      C, position = point_outside(points, size, position, O, r_squared)
      if not C:
        return diameter, r_squared                    # AB is solution

      # try finding another, more distant outside point
      D, position = point_outside(points, size, position, C, 4*r_squared)
      if not D:
        position = (position-1)%size
        points, size = eliminate(points, position, size-1)     # eliminate C
        if position == size: position = 0
        continue

      diameter = [C, D]
```