NISTIR 5831

An Algorithm for Computing the Minimum Covering Sphere in Any Dimension

Theodore H. Hopp
Charles P. Reeve

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Manufacturing Engineering Laboratory
Manufacturing Systems Integration Division
Gaithersburg, MD 20899-0001

NISTIR 5831

An Algorithm for Computing the Minimum Covering Sphere in Any Dimension

Theodore H. Hopp
Charles P. Reeve

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Manufacturing Engineering Laboratory
Manufacturing Systems Integration Division
Gaithersburg, MD 20899-0001

May, 1996

An Algorithm for Computing the Minimum Covering Sphere in Any Dimension

Theodore H. Hopp Charles P. Reeve[1]
National Institute of Standards and Technology
Gaithersburg, MD 20899-0001

Abstract

An algorithm is presented for computing the minimum covering sphere for a set of n points in d-dimensional space ($0 < n, d < \infty$). The steps of the geometric construction can readily be programmed for a computer. In the worst case, with all the points near the sphere surface, the expected computing time is estimated at $O(nd^{2.3})$.

Key words: algorithm; computational geometry; covering sphere; minimax fit; minimum covering sphere; surface fitting

1 Introduction

One tool needed in automated manufacturing work at the National Institute of Standards and Technology (NIST) is an algorithm for computing the sphere of minimum radius covering a set of points in three dimensions. The equivalent problem in two dimensions was first posed by Sylvester [11] in 1857. Recent publications by Megiddo [6], Dyer [1], and Preparata and Shamos [8] also address the two-dimensional case. Lawson [5] gives a compact iterative algorithm for solving the three-dimensional case. It is appealing in its simplicity, but converges too slowly for practical use when many of the points are near the surface of the sphere. Supowit [10] goes so far as to say "…for d [dimension] ≥ 3, no decent techniques are yet known." Elzinga and Hearn [2] consider the problem in n dimensions. They formulate it as a convex programming problem and give a finite decomposition algorithm based on the simplex method of quadratic programming.

In the above references the terms *smallest enclosing circle*, *smallest enclosing ball*, *smallest enclosing sphere*, and *minimum covering sphere* are used in the same context. In this paper we will use the latter term exclusively.

In searching for an efficient solution to the three-dimensional problem, we developed an algorithm for *geometrically* constructing the minimum covering sphere. Later we found that the algorithm extends easily into higher dimension.

2 The Problem

Let P be the set of n points $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$ in d-dimensional Euclidean space, E^d. (Vectors will be indicated by bold lower case and matrices by bold upper case.) For simplicity the word *sphere* will refer to the locus of points a fixed distance r from a given point $\mathbf{c}$, the center of the sphere, *regardless of the dimension*. Any sphere that encloses a set of points will be called a *covering sphere* (CS) for that set of points. The sphere of smallest radius that encloses a set of points will be called the *minimum covering sphere* (MCS) for the set of points. Mathematically the MCS problem is

$$\min_{\mathbf{c}} \max_{i} \|\mathbf{p}_i - \mathbf{c}\|$$

[1]Current address: Westinghouse Savannah River Company, Aiken SC 29808

where $\|\cdot\|$ is the Euclidean norm in E^d.

Two useful lemmas about the MCS, taken from [2], are re-stated here using our notation:

1) The center of the MCS can be expressed as a convex combination of at most d+1 of the given points.

2) The MCS exists and is unique.

When $n \geq 2$ and all the points are distinct, at least two and at most min(n,d+1) of these points define the MCS for the n points; thus there are

$$N = \sum_{i=2}^{\min(n,d+1)} \frac{n!}{i!(n-i)!} \tag{1}$$

combinations of points to be considered. Although the MCS is unique, the set of points defining the MCS is not necessarily unique. For example, the eight points (±1, ±1, ±1) all lie on a sphere centered at the origin with radius $\sqrt{3}$. The pair of points (1, 1, 1) and (-1, -1, -1) define the MCS, as do several other combinations of points. We consider this ambiguity unimportant and seek only the center and radius of the MCS. Some authors recommend that a first step in computing the MCS be to construct the convex hull of the n points and delete all points that are interior to the hull. We did not incorporate that step because we believe that the MCS problem is probably no more complex than the convex hull problem, and that our algorithm algorithm is as efficient as any convex hull algorithm for arbitrary dimension.

3 The Algorithm

Our algorithm geometrically constructs the MCS using an iterated two-step procedure. At the beginning of each iteration, a non-empty set $Q \subseteq P$ exists such that all points in Q are affinely independent. Also, a CS for P exists such that each point in Q lies on the surface of the CS. (The CS is not necessarily the MCS for Q.) Let **c** be the center of the CS. The iterated steps in constructing the MCS are:

1) Compute the center **t** of the MCS for Q, then remove from Q any points which do not constrain the MCS. (Point **t** then serves as a target.)

2) Shrink the CS by moving the center **c** toward **t** while maintaining all points in Q on the sphere surface. If the surface of the shrinking CS contacts a point in P but not in Q, fix **c** and add the new point to Q.

Initially **c** may be taken as point $\mathbf{p}_1$ and the set Q simply as the point farthest from $\mathbf{p}_1$. The MCS for P has been found when either:

a) Q contains exactly d+1 points at the end of step 1 (in which case **c** and **t** will coincide at the start of step 2), or

b) the center **c** of the shrinking CS in step 2 reaches target **t** without the sphere surface contacting a new point.

Points in Q before each iteration are called *candidate points*, and points in Q after the final iteration are called *constraining points*.

Figure 1 shows these iterations graphically for a small set of two-dimensional data. After each iteration the candidate points are shown as solid dots. The current CS is shown by the solid circle, and the MCS for the candidate points is shown by the broken circle. The algorithm terminates after the third iteration when step 1 ends with three candidate points.

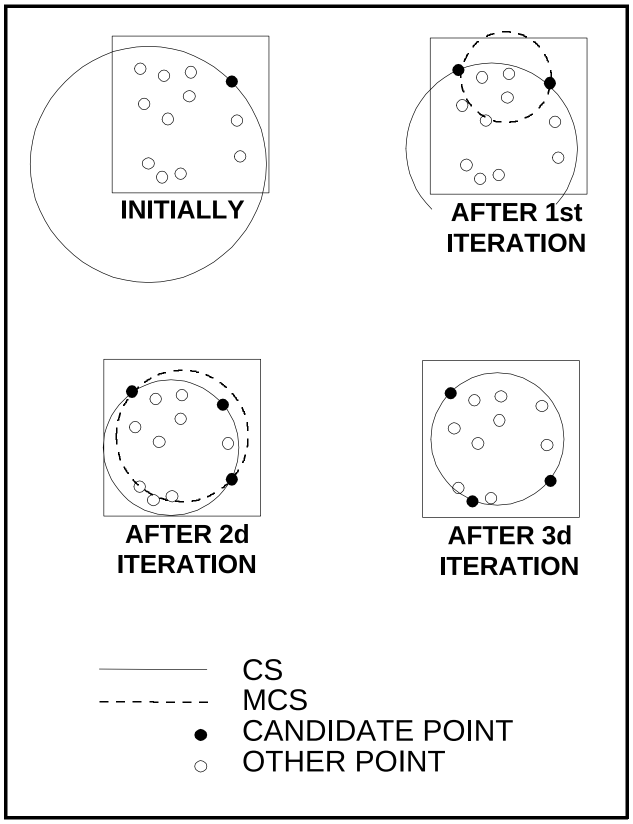

Figure 1 An illustration of the algorithm working in two dimensions.

Step 1 of the algorithm is accomplished as follows. To avoid double-subscripting the **p**'s, let the candidate points currently in Q be designated $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m$ where $0<m\leq d+1$. We first compute a point **t** which is equidistant from each point in Q and which lies in the affine hull of those points (see Hilton [4]). This point **t**, if it exists, will be unique. We first express **t** as

$$\mathbf{t} = \sum_{k=1}^{m} \lambda_k \mathbf{q}_k \tag{2}$$

Imposing the constraint

$$\sum_{k=1}^{m} \lambda_k = 1 \tag{3}$$

forces **t** to lie in the affine hull of the **q**'s. The constraints

$$(\mathbf{q}_j - \mathbf{q}_m)\cdot\left(\mathbf{t} - \frac{\mathbf{q}_j + \mathbf{q}_m}{2}\right) = 0 \tag{4}$$

for j=1, 2, …, m-1 force **t** to be equidistant from the **q**'s. Combining (2) and (3),

$$\mathbf{t} = \mathbf{q}_m + \sum_{k=1}^{m-1} \lambda_k(\mathbf{q}_k - \mathbf{q}_m). \tag{5}$$

Substituting (5) into (4) yields the linear system

$$\mathbf{A}\boldsymbol{\lambda} = \mathbf{b} \tag{6}$$

where **A** is an $(m\text{-}1)\times(m\text{-}1)$ matrix with elements

$$A_{jk} = (\mathbf{q}_j - \mathbf{q}_m)\cdot(\mathbf{q}_k - \mathbf{q}_m),$$

b is an $(m\text{-}1)$-vector with elements

$$b_j = A_{jj}/2$$

where j=1, 2, …, m-1, and

$$\boldsymbol{\lambda} = (\lambda_1\ \lambda_2\ \ldots\ \lambda_{m-1})^T.$$

After (6) is solved for the first m-1 λ's, λ_m is computed by (3). Because the points in Q are affinely independent, the m-1 vectors $\mathbf{q}_j$-$\mathbf{q}_m$ (j=1, 2, …, m-1) are linearly independent and (6) always has a unique solution. Because **A** is symmetric and positive definite, the Cholesky factorization may thus be used to efficiently solve (6). When d is large a significant savings in computing time can result.

If each of the m λ's is non-negative then the center of the sphere passing through the points is *inside* the convex hull of the points; therefore the sphere is the MCS for the points in Q.

If one or more λ's are negative then the center of the sphere is *outside* the convex hull of the points; therefore the sphere is not the MCS for the points in Q. It follows that one or more of the points in Q does not constrain the MCS and, in fact, lies *inside* it. We were unable to find an analytical method for identifying such points based solely on the computed λ's. We did, however, discover the following heuristic which works well: *Remove from Q the point corresponding to the most negative λ, reduce m by one, and re-solve (6) for the reduced set of λ's.* When all λ's are non-negative the center **t** is computed by (2), and step 1 is completed. (See Section 5 for a discussion of this heuristic.)

Step 2 of the algorithm is accomplished as follows. First note that the unique line passing through the non-identical points **c** and **t** has a parametric representation **c**+ρ(**t**-**c**) where $-\infty<\rho<\infty$. Points **c** and **t** are represented by ρ=0 and ρ=1 respectively. As noted earlier, the CS shrinks by moving its center from **c** to **t** along this line. For every point $\mathbf{p}_i$ such that $(\mathbf{t}-\mathbf{c})\cdot(\mathbf{q}_1-\mathbf{p}_i)\neq 0$, there is a finite ρ_i such that $\mathbf{c}+\rho_i(\mathbf{t}-\mathbf{c})$ is equidistant from $\mathbf{q}_1$ and $\mathbf{p}_i$; that is,

$$\left[\mathbf{c}+\rho_i(\mathbf{t}-\mathbf{c})-\frac{\mathbf{q}_1+\mathbf{p}_i}{2}\right]\cdot(\mathbf{q}_1-\mathbf{p}_i) = 0. \qquad (7)$$

Figure 2 illustrates, in two dimensions, how $\mathbf{p}_i$ falls into one of four regions depending on its corresponding ρ_i value. In higher dimension the diameter of the MCS indicated by the dashed line can be thought of as a hyperplane normal to the vector **t**-**c**. When $\rho_i \leq 0$, $\mathbf{p}_i$ is in region 1; when $\rho_i \geq 1$, $\mathbf{p}_i$ is in region 2; when $0 \leq \rho_i < 1$, $\mathbf{p}_i$ is in region 3; when ρ_i does not exist, $\mathbf{p}_i$ is in region 4 (the hyperplane containing $\mathbf{q}_1$ and normal to **t**-**c**). As **c** moves toward **t**, the shrinking CS always includes regions 1, 2, and 4; therefore only points in region 3 are candidates to constrain the shrinking CS. (For instance, the shrinking CS will "contact" **p3** when the center reaches **c3**.) Furthermore, the first point that the CS contacts is the one with the smallest non-negative ρ value, ρ_{min}. When no points are in region 3, the center of the CS is free to move all the way to **t** (in effect, ρ_{min}=1) and we are done. Step 2 can thus be stated:

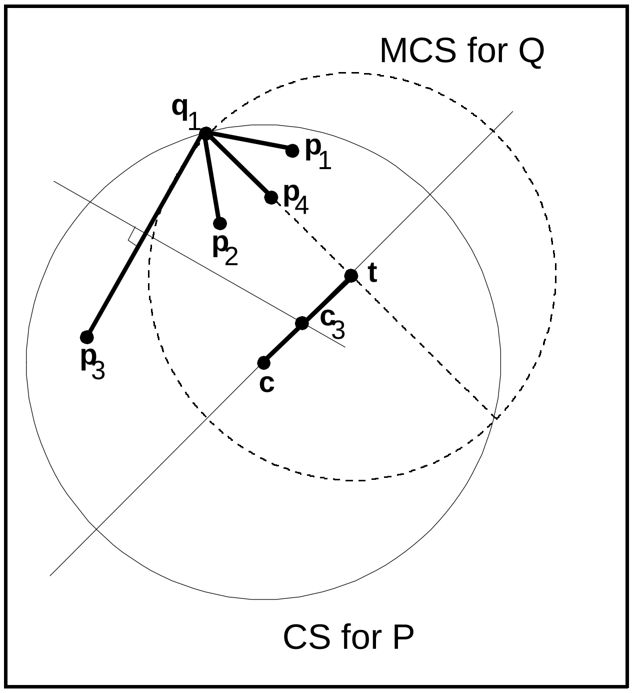

Figure 2 The geometry used to find the next candidate point.

a) For each $\mathbf{p}_i$ contained in Q set ρ_i=1. For each $\mathbf{p}_i$ not in Q compute

$$d_i = (\mathbf{t}-\mathbf{c})\cdot(\mathbf{q}_1-\mathbf{p}_i).$$

If $d_i \le 0$ then $\mathbf{p}_i$ cannot be the next candidate point, so set ρ_i=1. If $d_i > 0$ then compute

$$\rho_i = \left(\frac{\mathbf{q}_1+\mathbf{p}_i}{2}-\mathbf{c}\right)\cdot\frac{\mathbf{q}_1-\mathbf{p}_i}{d_i}$$

from (7).

b) Compute $\rho_{min} = \min\{\rho_i | i=1, 2, \ldots, n\}$. If $\rho_{min}<1$ then the point corresponding to ρ_{min} is the next candidate point and is added to Q. The center of the new CS is computed to be

$$\mathbf{c}^{[new]} = \mathbf{c}^{[old]} + \rho_{min}(\mathbf{t}-\mathbf{c}^{[old]}).$$

If ρ_{min}=1 then the MCS for the current set Q is the final MCS, centered at $\mathbf{t}$, for all n points.

Note in Figure 1 that, as the iterations progress, the solid circles get smaller and the broken circles get larger. This behavior, which occurs regardless of the dimension, is the trademark of our algorithm. More formally stated, the sequence of radii of the *minimum* spheres covering set Q is strictly increasing, and the sequence of radii of the *covering* spheres for set P is strictly decreasing. When the sequences reach a common value the *minimum covering* sphere has been found.

4 Timing Test Results

Timing tests were performed on a CDC Cyber 180/855 computer at the National Institute of Standards and Technology. The computations were done using $\approx$14 decimal digit arithmetic. Two types of artificial data sets were generated using pseudo-random number generators. One type consists of points uniformly distributed throughout the unit sphere. These data were designated *uniform* points. The other type consisted of points uniformly distributed in the spherical shell of width 10^{-6} at the surface of the unit sphere. These data were designated *surface* points. Both types of data were obtained by first generating uniformly-distributed points on the surface of a d-dimensional unit sphere by the method of Muller [7]. These points were then scaled to the interval $(r,1)$ by the random variable $R=[U+(1-U)r^d]^{1/d}$ where U is a random variable with uniform(0,1) distribution. For the *uniform* points r was 0, and for the *surface* points r was $1-10^{-6}$. The *surface* points were meant to simulate the most demanding case for the algorithm. In E^3 such points might result from measurements on the surface of a ball with a high-precision coordinate measuring machine. The *uniform* points were analyzed to provide a contrast.

In each case random data sets with selected values of n as high as 1280 and d as high as 32 were analyzed, keeping track of the central processing unit (CPU) times. In the *uniform* case the expected CPU time was estimated to be $O(n^{1.1}d^2)$, and in the *surface* case the expected CPU time was estimated to be $O(nd^{2.3})$. (Estimates of the exponents were obtained by linear least squares using the log of the model $cputime = \gamma n^{\alpha} d^{\beta}$).

Table 1 Algorithm performance on two types of data.

		Uniform points		*Surface* points	
d	***n***	Range[*] (iterations)	Range[*] (CPU sec.)	Range[*] (iterations)	Range[*] (CPU sec.)
2	10	3-4	.001-.002	3-5	.002-.003
2	100	3-8	.009-.027	6-11	.020-.036
2	1000	4-7	.122-.229	10-15	.336-.501
4	10	3-6	.002-.005	4-9	.003-.011
4	100	7-12	.044-.073	13-17	.079-.104
4	1000	10-169	.527-.884	17-31	.961-1.76
8	10	5-8	.008-.014	6-10	.009-.022
8	100	13-22	.154-.259	25-44	.305-.566
8	1000	19-33	1.87-3.38	40-74	4.22-7.89
16	10	6-10	.016-.037	8-11	.025-.047
16	100	25-35	.622-.921	55-78	1.91-2.75
16	1000	49-72	9.72-14.3	94-118	19.4-24.5

[*]based on ten replications

For each of the *d* and *n* values shown in Table 1, ten replicate data sets were generated and analyzed. The ranges of observed CPU times and number of iterations are shown. We regret that we were unable to find published timing data for other algorithms with which to compare our own.

In our simulations we have rarely encountered endless cycles. We have identified only one mechanism by which endless cycling occurs. It may happen when two points nearly tie to become the next candidate point during step 2. The failure occurs as follows. During one iteration, one of the two points—call it $\mathbf{p}_i$—is found to be the next candidate point and is added to Q. However, the other point $\mathbf{p}_j$ (not in Q) has a value of ρ only sightly higher than that of $\mathbf{p}_i$. During the next iteration, after the target center has changed, point $\mathbf{p}_j$ has a negative (but very small magnitude) computed ρ *but is not in region 1*. In consequence, the center must be pulled back from the target (rather than approach it) to bring $\mathbf{p}_j$ back into the CS. This may then return the configuration to where it was before the first iteration, and the cycle repeats. We have not analyzed this failure completely, but it seems to occur only when the algorithm has nearly converged, so stopping the iterations and returning the current covering sphere produces a nearly exact answer.

5 Discussion

We see three areas in which further research might lead to improvements in our algorithm. The first concerns the heuristic in step 1 for deleting points in Q when negative λ's are obtained. Our initial thought was to remove all points corresponding to negative λ's in order to reduce the size of linear system (6) as much as possible. This heuristic works in that the algorithm eventually iterates to the correct answer. However, in many cases points which do indeed define the MCS of the current set Q are removed from Q. They immediately re-enter Q in step 2, thus increasing the total number of iterations. After examining many sets of simulated data we came to the conclusion (but did not prove) that removing the single point with the most negative λ from Q, and recomputing (6), is the most efficient heuristic. In none of these test cases did this heuristic remove a point from the current set Q which actually defined the MCS of that Q. As in the above case, a failure of this heuristic would not result in a failure of the algorithm.

Another area for possible improvement is the addition of a rule for permanently eliminating non-constraining points. In two dimensions, for example, points eliminated from Q in step 1 and points in regions 1 and 2 in step 2 can safely be ignored during the rest of the execution of the algorithm. When n is large a considerable savings in computing can result, depending on the structure of the data. In three and higher dimensions, however, neither of these rules is valid. Since computing time increases much more rapidly with d than n, we decided not to incorporate the special rules for the case d=2. If an elimination rule can be found which works in any dimension and is easy to implement, it might greatly enhance the efficiency of the algorithm.

A third area for improvement is in choosing which point in Q should be designated $\mathbf{q}_m$ in step 1. Depending on that choice, the condition number of $\mathbf{A}$, and thus the accuracy with which the λ's are computed, can vary widely. In a brief study we found that letting $\mathbf{q}_m$ be the point nearest the mean of the $\mathbf{q}$'s, in the Euclidean sense, was optimal most of the time. On computer calculations with a relatively short word length, say six to eight digits, it may be worth the extra computational time to incorporate this or a similar heuristic. Excellent discussions of matrix condition numbers and the effects of rounding errors can be found in Golub and Van Loan [3] and in Stewart [9].

6 Conclusion

A simple and efficient algorithm for computing the minimum covering sphere in any dimension has been presented. The number of points (n) and dimension (d) are bounded above only by computer storage limitations. In infinite-precision arithmetic the MCS must always be found in at most N iterations, as given in Equation (1). However, the finite-precision arithmetic of a computer opens the possibility of endless cycling among several sets of points. To deal with this possibility, any computer software which performs this algorithm should incorporate an upper bound on the number of iterations.

Source code in FORTRAN or C for our implementation of the algorithm is available from the authors.

References

[1] M.E. Dyer, *Linear Time Algorithms for Two- and Three-variable Linear Programs*, SIAM Journal of Computing **13**(1) (1984), pp. 31-45.

[2] D.J. Elzinga and D.W. Hearn, *The Minimum Covering Sphere Problem*, Management Science **19**(1) (1972), pp. 96-104.

[3] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 1985.

[4] P. Hilton, *Advanced Topology, An Introductory Course*, Courant Institute of Mathematical Sciences, New York, 1969.

[5] C.L. Lawson, *The Smallest Covering Cone or Sphere*, SIAM Review **7**(3) (1965), pp. 415-417.

[6] N. Megiddo, *Linear Time Algorithm for Linear Programming in R^3 and Related Problems*, SIAM Journal of Computing **12**(4) (1983), pp. 759-776.

[7] M.E. Muller, *A Note on a Method for Generating Points Uniformly on N-Dimensional Spheres*, Communications of the ACM, **2** (1959), pp. 19-20.

[8] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[9] G.W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[10] K.J. Supowit, *Grid Heuristics for Some Geometric Covering Problems*, Advances in Computing Research: Computational Geometry—Volume 1, JAI Press, Inc., Greenwich, 1983, pp. 228-229.

[11] J.J. Sylvester, *On Poncelot's approximate Linear Valuation of Surd Forms*, Philosophical Magazine, Ser. 4, 20 (1860), pp. 203-222.