Some useful crystallographic algorithms

ECM29 Computing School

21st August 2015

George M. Sheldrick

http://shelx.uni-ac.gwdg.de/SHELX/

What is an algorithm ?

An algorithm is a way of calculating something that even the person who invented it cannot understand! Usually algorithms involve some clever mathematics to do something faster – sometimes by orders of magnitude – or 'better' than the obvious way of doing it.

Sort algorithms

Sorting algorithms are a highlight of many informatics courses because the difference in speed can be enormous. An obvious method such as scanning A(1..N) to find the largest element, then swapping it with A(1), then scanning A(2..N) and swapping the largest element with A(2), then scanning A(3..N) etc. takes a time of order N^2 (which for large N can be all week). The best general algorithms are of order NlogN but are relatively complicated (best to call a library routine).

Sometimes – as with sorting reflection lists – we can take advantage of the special features of the particular problem to reduce the order to almost N. For 10000 reflections, N^2 is 100000000, NlogN is 50000 but N is only 10000 (but the constant factor multiplying the order may differ)!

A typical sort algorithm

The following FORTRAN routine (called comb-sort) is a good general purpose algorithm for sorting a few hundred items (e.g. Fourier map peaks). Although not quite NlogN it is fast because it has a low overhead. The array A(1..N) (containing e.g. peak heights) is sorted into descending order. In practice the IF..ENDIF loop would also need to swap the atom coordinates x, y and z as well as the peak heights.

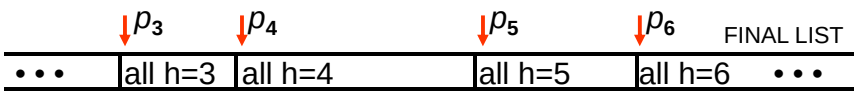
```
      K=N
1     K=INT(REAL(K)/1.2796)
      IF(K.LT.1)K=1
      IF(K.EQ.9.OR.K.EQ.10)K=11
      M=0
        DO 2 I=1,N-K
        J=I+K
        IF(A(J).GT.A(I))THEN
          Q=A(J)
          A(J)=A(I)
          A(I)=Q
          M=1
          ENDIF
2       CONTINUE
      IF(M+K.GT.1)GOTO 1
```

Sorting reflection lists (order N)

First *h,k,l* are transformed to a standard equivalent (e.g. maximum *l*, if *l* equal then maximum *k*, if both *k* and *l* equal then maximum *h*). Then the maximum and minimum values of each index are found.

To sort on h, scan list, count how often each h is present, storing the counts in an integer array $N(h_{min}..h_{max})$. This is then converted so that it holds pointers p_h to the final list: The list is scanned again, putting each reflection into the final location pointed to by p_h and then incrementing p_h.

The list is sorted first on *h*, then on *k*, and finally on *l*. In the final sorted list, equivalents finish next to each other and so can easily be averaged.



Calculation of interatomic distances

There are two general approaches:

(A) Convert to Cartesian coordinates, then use Pythagoras:

x, *y*, *z* (crystal fractional coordinates) $\longrightarrow$ *X*, *Y*, *Z* (orthogonal)

then $d^2 = (X_1–X_2)^2 + (Y_1–Y_2)^2 + (Z_1–Z_2)^2$

(B) Use crystal coordinates directly:

$$d^2 = a^2 (x_1–x_2)^2 + b^2 (y_1–y_2)^2 + c^2 (z_1–z_2)^2 + 2bc \cos\alpha\, (y_1–y_2)\, (z_1–z_2) + 2ac \cos\beta\, (x_1–x_2)\, (z_1–z_2) + 2ab \cos\gamma\, (x_1–x_2)\, (y_1–y_2)$$

Method (A) is simpler and usually faster, but with method (B) it is much easier to handle symmetry equivalent atoms (see tutorial).

Orthogonal coordinates

A large variety of transformations are in use. For example, to convert fractional coordinates x, y, z to *default PDB* orthogonal coordinates *X*, *Y*, *Z* the following transformation is used:

$$\begin{aligned} X &= ax + (b\cos\gamma)\, y + (c\cos\beta)\, z \\ Y &= 0 + (b\sin\gamma)\, y + (–c\sin\beta\cos\alpha^*)\, z \\ Z &= 0 + 0 + (c\sin\beta\sin\alpha^*)\, z \end{aligned}$$

where $\cos\alpha^* = (\cos\beta\cos\gamma – \cos\alpha) / (\sin\beta\sin\gamma)$

and $\sin\alpha^* = \sqrt{(1 – \cos^2\alpha^*)}$

Of course the coefficients are calculated once and stored; for the *X*, *Y*, *Z* $\longrightarrow$ *x*, *y*, *z* transformation the inverse matrix is used (given on the SCALE1, SCALE2 and SCALE3 PDB cards), or the same coefficients as above can be used (in the right order!):

$$\begin{aligned} z &= Z / (c\sin\beta\sin\alpha^*) \\ y &= (Y – (–c\sin\beta\cos\alpha^*)\, z) / (b\sin\gamma) \\ x &= (X – (b\cos\gamma)\, y – (c\cos\beta)\, z) / a \end{aligned}$$

Finding nearest neighbors

At first sight we need to test against all *M* symmetry equivalents and all 26 surrounding unit-cells, so to find all short distances (e.g. bonds) involving all N atoms in a structure requires calculating $13MN(N–1)$ interatomic distances, which could be slow! The following algorithm is faster:

First prepare a separate list of all the atoms *x', y', z'* in one unit cell, taking symmetry and lattice type into account, adding 99.5 to all *x', y'* and *z'* values (!)

Then find all distances with $d^2 < d_{min}^2$ using:

$\Delta x = (x'–x) \bmod 1 – 0.5$

$\Delta y = (y'–y) \bmod 1 – 0.5$

$\Delta z = (z'–z) \bmod 1 – 0.5$

Note: in Fortran, T–AINT(T) can be appreciably faster than AMOD(T,1.0)

and: $d^2 = a^2\Delta x^2 + b^2\Delta y^2 + \ldots + 2ab\cos\gamma\,\Delta x\,\Delta y$

The coordinates of the neighboring atoms *x', y'* and *z'* may be found using:

$x'' = x + \Delta x$, $y'' = y + \Delta y$, $z'' = z + \Delta z$

Speeding it up

For large structures this algorithm can be speeded up by another couple of orders of magnitude!

First divide the structure into groups of say 10 atoms that are expected to be close to one another (in macromolecules one can conveniently group atoms with the same residue number). For each group the mid-point $\overline{x}, \overline{y}, \overline{z}$ and the maximum distance R of any atom from the midpoint are found.

The distance search is first applied to the mid-points. Only for the rare cases when a pair of mid-points i and j are within $R_i + R_j + d_{min}$ of one another do the individual atoms in the two groups need to be tested against one another.

Special positions

Special positions may be found in a similar way to the search for short distances, but the inner loop is over the equivalents of the current atom, not the complete unit-cell, so it is much faster.

If d^2 < (say) 0.1 Å^2 one could consider the atom to be on a special position (but see the SPEC instruction in SHELXL!). The occupancy for SHELXL is given by the reciprocal of the number of atoms that coalesce to a single atom.

To idealize the coordinates so that the atom lies exactly on the special position, simply average all x' values, all y' values and all z' values, including the original x, y and z. It is advisable to iterate this calculation about three times because for peaks some distance from the special position the above test may fail for some of the equivalent positions in the first pass.

The shortest distance matrix

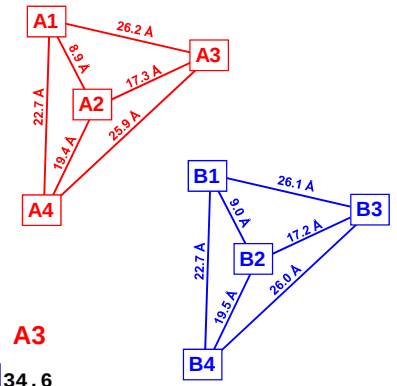
The shortest distance matrix (SDM) for N unique atoms is a NxN matrix in which each location stores the shortest distance d_{ij} between the atoms I and j taking symmetry into account. To find these distances it is sufficient to consider one position for atom i and all equivalents for atom j. Since the matrix is symmetrical it can also be displayed in triangular form. The diagonal elements are usually calculated as the shortest non-zero distance between equivalents of the same atom.

Despite the drastic simplification involved – all other distances are ignored – this matrix is rather useful!

At the same time as generating all possible unique vectors between two atoms, one can also make a list of the Patterson values at the vectors. This list is then sorted and the values for (say) the weakest 30% summed (a *Patterson minimum function, PMF*). The combination of PMF values and shortest distances is very useful for Patterson interpretation, and may be found in the .lst file from SHELXD when the PATS instruction is invoked.

Identifying NCS for JIA SeMet MAD

Peak	self	cross-vectors							
99.9	37.9 19.1	A1							
93.9	54.9 19.1	8.9 18.6	A2						
93.6	26.3 19.1	22.7 15.0	19.4 17.1	A4					
91.2	48.7 18.4	21.9 36.0	21.4 20.0	20.6 20.4	B2				
85.2	31.0 13.5	23.9 21.9	20.8 21.2	32.5 19.3	19.5 15.3	B4			
83.3	38.0 28.2	26.2 24.3	17.3 19.1	25.9 17.8	30.0 19.3	23.5 20.8	A3		
76.6	37.5 14.6	18.5 15.7	21.7 14.5	23.6 12.6	9.0 18.1	22.7 14.9	34.6 20.9	B1	
69.1	29.8 14.0	34.7 19.8	30.0 20.9	23.2 19.8	17.2 9.6	26.0 16.7	28.7 18.2	26.1 19.4	B3
22.3	54.6 0.0	10.3 2.7	4.2 0.0	18.4 2.8	17.6 11.5	17.7 0.0	17.4 0.0	18.8 2.1	26.3 0.0



Thomas R. Schneider

Bringing the atoms together in SHELXT

The following algorithm used in **SHELXT** does not require that elements and hence covalent radii are correctly assigned to the atoms!

1. Generate the SDM (Shortest Distance Matrix – shortest distances between unique atoms, taking symmetry into account).
2. Set a flag to −1 for each unique atom, then change it to +1 for one atom - it does not matter which.
3. Search the SDM for the shortest distance for which the product of the two flags is −1; *if none, exit*.
4. Symmetry transform the atom with flag −1 for this distance so that it is as close as possible to the atom with flag +1, then change its flag to +1.
5. GOTO 3

This diabolically simple algorithm not only builds the molecules as we would intuitively expect them whatever the space group, but also clusters them in a chemically sensible way, making the structure instantly recognizable.

Application to macromolecules

The shortest distance matrix is probably not the optimal method to cluster macromolecules. A better approach might be to prepare a similar matrix for molecules instead of atoms, and to store in it the largest contact areas between two molecules instead of the shortest distances, but handling symmetry in the same way.

Tutorial

The file insulin.pdb is a highly mutilated (but legal) PDB file from a high resolution insulin structure. As would be the case for the initial atom sites from a structure solution, some atoms need to be symmetry transformed to show the structure properly. You should write a program (any programming or scripting language may be used) that reads this file in order to:

1. Generate crystal coordinates for the atoms by multiplying the orthogonal coordinates in the file by SCALE1 etc. (see next page).

2. Set up a triangular SDM (matrix of shortest distances between unique atoms) taking the space group symmetry and lattice centering into account. The distances along the diagonal should be the shortest distances between an atom and all its symmetry equivalents.

3. Inspect this matrix to deduce which atoms lie on special positions and which sulfurs make disulfide bonds.

The structure contains two monomers in the asymmetric unit, each with 3 disulfides and one Zn. Suggest an algorithm to divide the structure into these monomers (more difficult).

Further information for the tutorial

Space group R3 on hexagonal axes requires the following symmetry operations and lattice translations:

x, y, z	**0, 0, 0**
-y, x-y, z	**⅔, ⅓, ⅓**
y-x, -x, z	**⅓, ⅔, ⅔**

The SCALEn cards (SCALE1 s_{11} s_{12} s_{13} t_1 etc) are used to convert the orthogonal coordinates ($\mathbf{x}_o$) to crystal coordinates **x** as follows:

$$\mathbf{x} = s_{11}*x_o + s_{12}*y_o + s_{13}*z_o + t_1$$

$$\mathbf{y} = s_{21}*x_o + s_{22}*y_o + s_{23}*z_o + t_2$$

$$\mathbf{z} = s_{31}*x_o + s_{32}*y_o + s_{33}*z_o + t_3$$

Note: the PDB file contains the chain and residue numbers for checking the results against PDB entry 1MSO, your program should ignore these!