

# Some computational algorithms used in SHELX

IUCr Montreal, 12<sup>th</sup> August 2014

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.

### Examples in this tutorial

Sorting reflection lists

Fast calculation of interatomic distances

Distance matrices

Unique molecule assembly

Clustering macromolecules

PART numbers in SHELX

## Sort algorithms

Sorting algorithms are the 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  $N\log N$  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  $N$ . For 10000 reflections,  $N^2$  is 100000000,  $N\log N$  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  $N\log N$  it is fast because it has a low overhead. The array

```
      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
```

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.

## 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 results in an integer array  $N(h_{\min}..h_{\max})$ . This is then converted so that it holds pointers  $\rho_h$  to the final list: The list is scanned again, putting each reflection into the final location pointed to by  $\rho_h$  and then incrementing  $\rho_h$ .

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.

	↓ $\rho_3$	↓ $\rho_4$	↓ $\rho_5$	↓ $\rho_6$	FINAL LIST
•••	all h=3	all h=4	all h=5	all h=6	•••

## 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.

## 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:  $d^2 = a^2 \Delta x^2 + b^2 \Delta y^2 + \dots + 2ab \cos \gamma \Delta x \Delta y$

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

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

## 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  $\bar{x}, \bar{y}, \bar{z}$  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  $d^2 < (\text{say}) 0.1 \text{ \AA}^2$  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  $z$  too. 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.

Duplicates caused by special positions need to be eliminated from the list of atoms filling one unit-cell before searching for neighbors, but care has to be taken when cations share sites etc.

## The shortest distance matrix

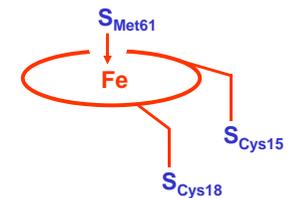
The shortest distance matrix (SDM) for  $N$  unique atoms is a  $N \times N$  matrix in which each location stores the shortest distance  $d_{ij}$  between the atoms  $i$  and  $j$  taking symmetry into account. To find these distances it is sufficient to consider one position for atom  $i$  and all 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.

## Cytochrome c6 in space group R3

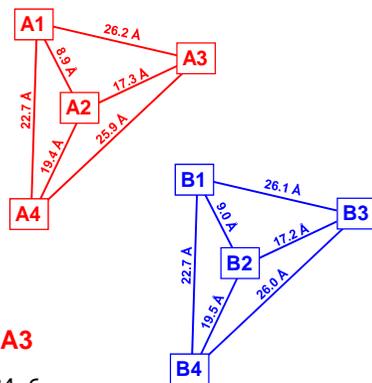
x	y	z	self	cross-vectors								
0.826	0.212	0.173	19.25 27.7	<b>Fe</b>								
0.907	0.218	0.305	18.94 6.8	6.83 11.8	<b>S<sub>Cys</sub></b>							
0.805	0.217	0.210	21.34 0.4	1.61 9.4	6.16 0.0							
0.818	0.231	0.246	22.02 1.8	3.13 10.3	4.62 1.7	1.83 0.0						
0.974	0.130	0.115	7.77 4.8	6.63 10.6	8.87 1.6	7.86 0.0	8.51 0.0	<b>S<sub>Cys</sub></b>				
0.941	0.119	0.142	9.97 0.0	5.59 7.3	8.10 0.0	6.69 0.0	7.40 0.0	1.66 0.0				
0.048	0.173	0.062	6.24 2.5	9.36 4.9	10.11 0.1	10.66 0.0	11.10 0.0	3.86 0.0	5.48 0.0			
0.982	0.223	0.974	12.80 0.0	9.36 7.5	10.45 1.2	10.75 0.0	11.07 0.0	6.28 0.0	6.03 0.0	4.76 0.0		
0.812	0.273	0.149	21.58 8.8	2.50 12.9	7.80 3.5	3.07 0.0	4.05 0.0	8.09 1.6	7.43 0.0	10.01 0.0	9.22 0.0	<b>S<sub>Met</sub></b>



A  $\sqrt{(E^3F)}$  super-sharp Patterson was used to derive this table

## Crossword table for JIA SeMet MAD

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

## PART numbers in SHELXL

In order to be able to generate restraints and place hydrogen atoms etc., SHELX sets up a connectivity table that defines which atoms are *bonded*. Overlapping alternative conformations may be defined by different PART numbers (similar to the alternative location indicator in PDB format). Atoms can bond only if the distance between them is short enough and either:

They have the same PART number, or

One of the two PART numbers is zero (the default).

This fails if there is disorder within a disorder, e.g. a n-alkyl side-chain splits at C2 and again at C4. Atoms up to and including C2 are in PART 0, C3A and C4A in PART 1, C3B and C4B in PART 2, but where are C5A..D?

The answer is that C5A..D are in PARTs 3..6 and the new instructions:

```
BIND 1 3
BIND 1 4
BIND 2 5
BIND 2 6
```

extend the PART rules.