Commission on Crystallographic Computing
International Union of Crystallography
http://www.iucr.org/iucr-top/comm/ccom/
Newsletter No. 4, August 2004

This issue's theme:
"Restraints, Constraints and using extra observables"
http://www.iucr.org/iucr-top/comm/ccom/newsletters/

Table of Contents

(This Issue's Editor: Lachlan Cranswick)

(Editor's warning – unless you want to kill 103 pages worth of forest – DO NOT press the "print" button. For hardcopies – you may like to only print out the articles of personal interest.)

CompComm Chairman’s Message

This is already the fourth newsletter of the IUCr computing commission, capably put together by our editor, Lachlan Cranswick, with articles of interest on this issue’s main topic: 'Constraint & Restraint refinement'. The subject is of course not new. However, from the current users point of view, what counts is whether those features are available within their software environment of choice (SHELXL, CRYSTALS, SIR, ..) and well explained. Hopefully, these article may also lead to more 'looking over the fence'.

The historical article by Dick van der Helm on early computing brought back to me the many nights (in the 60’s) spent at the computer centre in Utrecht writing Direct Methods programs and doing 3D maps and SFLS calculations within the constraints of 16K 27bit words. I still have the sound of the lineprinter in my ear that directly reflected the first, second and third Fourier summation steps.

There are 9 computing related sessions planned for the Florence 2005 IUCr congress. The program for the IUCr computing school to be held in Siena immediately prior to the Florence meeting is in its refinement stage. More info will be available soon, and published in the next edition of this newsletter. The upcoming ECM-22 Budapest meeting (25-31 Aug 2004) includes 5 computing related sessions :

- Crystal Structure validation: challenges and tools (Chair/Co-chair: Tony Linden/Richard Cooper)
- Advances and pitfalls in automated structure determination (Chair/Co-Chair: Ton Spek/Simon Parsons)
- Crystallographic graphics tools and user interfaces(Co-chair: Martin Noble/Laszlo Parkanyi)
- Advances in powder diffraction methods (Chair/Co-chair: Jordi Rius/Eric Mittemeijer; Jointly with the Powder SIG)
- New methods for phasing, model building and real-time refinement (Chair/Co-chair: Eleanor Dodson/George Sheldrick; Jointly with the Protein SIG)

Ton Spek, Chairman or the IUCr Computing Commission, (a.l.spek@chem.uu.nl)

From the Editor of Newsletter No. 4

This current newsletter contains a variety of articles on the theme of **"Restraints, Constraints and using extra observables"**. The alleged first documented case of the crystallographer correcting the preconceived incorrect molecular restraints of the chemist is that of Mills and Nyburg, "The Molecular Structure of Aspidospermine", *Tetrahedron Letters* No, 11 pp 1-3, 1959; and Mills and Nyburg, *J. Chem. Soc.*, pp 1458-1463 (1960) (this is inaccurately described in "From Classical to Modern Chemistry: The Instrumental Revolution", Royal Society of Chemistry, 2002, ISBN 0-85404-479-5). However, if diffraction data does not allow for a complete analysis, extra observables and chemical information, when intelligently applied, can be the difference between success and failure in solving crystallographic problems. As with most of crystallography, without good algorithms and computer code, the attempt to use extra observables would be a frustrating business. While a goodly range of articles was sought for this newsletter, it is not exhaustive, and this theme will no-doubt have to be revisited in a future edition.

The next issue (due January 2005) is intended to have the theme of **"At Right Angles to Conventional Crystallographic reality: computing and algorithms related to incommensurate, quasicrystal, pair distribution function and magnetic structures"**. That some structures don't fit nicely into a convenient commensurate unit-cell can be a disturbing thought, and highly inconvenient, to the classical crystallographer (similar perhaps to those who were more at home with classical physics vs that of the quantum world). A possible future requirement in crystallography may be the revisiting of published crystal structures where more exhaustive examination with CCD single crystal diffractometers may show weak satellite reflections or diffuse scattering requiring proper explanation. Submissions relating to the control and visualization of raw single crystal image data for the elucidation of many of the above types of structural problems are encouraged. Articles of a general nature outside the above theme are also welcome, especially those dealing with the history of crystallographic computing.

Lachlan Cranswick (Lachlan.cranswick@nrc.gc.ca)

The IUCr Commission on Crystallographic Computing - Triennium 2003-2005

Chairman: Prof. Dr. Anthony L. Spek
Director of National Single Crystal Service Facility,
Utrecht University,
H.R. Kruytgebouw, N-801,
Padualaan 8, 3584 CH Utrecht,
the Netherlands.
Tel: +31-30-2532538
Fax: +31-30-2533940
E-mail: a.l.spek@chem.uu.nl
WWW: http://www.cryst.chem.uu.nl/spea.html

Professor I. David Brown
Brockhouse Institute for Materials Research,
McMaster University,
Hamilton, Ontario, Canada
Tel: 1-(905)-525-9140 ext 24710
Fax: 1-(905)-521-2773
E-mail: idbrown@mcmaster.ca
WWW: http://www.physics.mcmaster.ca/people/faculty/Brown_ID.html

Lachlan M. D. Cranswick
Neutron Program for Materials Research (NPMR),
National Research Council (NRC),
Building 459, Station 18, Chalk River Laboratories,
Chalk River, Ontario, Canada, K0J 1J0
Tel: (613) 584-8811 ext: 3719
Fax: (613) 584-4040
E-mail: lachlan.cranswick@nrc.gc.ca
WWW: http://neutron.nrc.gc.ca/peep.html#cranswick

Dr Vincent Favre-Nicolin
CEA Grenoble
DRFMC/SP2M/Nano-structures et Rayonnement Synchrotron
17, rue des Martyrs 38054 Grenoble Cedex 9
38054 Grenoble Cedex 9 – France
Tel: (+33) 4 38 78 95 40
Fax: (+33) 4 38 78 51 97
E-mail: vincefn@users.sourceforge.net
WWW: http://objcryst.sourceforge.net/

Dr Ralf Grosse-Kunstleve
Lawrence Berkeley Lab
1 Cyclotron Road,
BLDG 4R0230,
Berkeley, CA 94720-8235, USA.
Tel: 510-486-5713
Fax: 510-486-5909
E-mail: rwgk@yahoo.com
WWW: http://cci.lbl.gov/

Prof Alessandro Gualtieri
Università di Modena e Reggio Emilia,
Dipartimento di Scienze della Terra,
Via S.Eufemia, 19,
41100 Modena, Italy
Tel: +39-059-2055810
Fax: +39-059-2055887
E-mail: alex@unimore.it
WWW: http://www.terra.unimo.it/mineralogia/gualtieri.html

Prof Ethan A Merritt
Department of Biological Structure
University of Washington
Box 357420, HSB G-514
Seattle, Washington, USA
Tel: 206 543 1861
Fax: 206 543 1524
E-mail: merritt@u.washington.edu
WWW: http://www.bmsc.washington.edu/people/merritt/

Dr. Simon Parsons
School of Chemistry
Joseph Black Building,
West Mains Road,
Edinburgh, Scotland EH9 3JJ, UK
Tel: +44 131 650 5804
Fax: +44 131 650 4743
E-mail: s.parsons@ed.ac.uk
WWW: http://www.chem.ed.ac.uk/staff/parsons.html

Dr. Bev Vincent
RigakuMSC
9009 New Trails Dr,
The Woodlands, Texas 77381-5209, USA
Tel: 281-363-1033
Fax: 281-364-3628
E-mail: brv@RigakuMSC.com
WWW: http://www.rigakumsc.com/

Consultants

Dr David Watkin
Chemical Crystallography,
Oxford University,
9 Parks Road,
Oxford, OX1 3PD, UK.
Tel: +44 (0) 1865 272600
Fax: +44 (0) 1865 272699
E-mail: david.watkin@chemistry.oxford.ac.uk
WWW: http://www.chem.ox.ac.uk/researchguide/djwatkin.html

Dr Harry Powell
MRC Laboratory of Molecular Biology,
Hills Road, Cambridge, CB2 2QH, UK.
Tel: +44 (0) 1223 248011
Fax: +44 (0) 1223 213556
E-mail: harry@mrc-lmb.cam.ac.uk
WWW: http://www.mrc-lmb.cam.ac.uk/harry/

A preliminary announcement of the IUCr Commission on Crystallographic Computing

IUCr Commission on Crystallographic Computing

Siena 2005: Crystallographic Computing School

Certosa di Pontignano, University of Siena, Italy 18th - 23rd August 2005

(just prior to the Florence IUCr 2005 congress)

http://www.iucr.org/iucr-top/comm/ccom/siena2005/

School Organisers: Prof Anthony Spek (Utrecht), Prof. Marcello Mellini (Siena), Prof. Alessandro Gualtieri (Modena), Dr Harry Powell (Cambridge), Lachlan Cranswick (NRC Chalk River)
Consultants: Dr David Watkin (Oxford), Dr Simon Parsons (Edinburgh)

Siena, Tuscany
ROME
Florence
Milan
Turin
Verona
Trieste
Venice
Genoa
Bologna
La Spezia
Livorno
Naples
Salerno
Bari
Brindisi
Taranto
Palermo
Messina
Reggio di Calabria
Catania
Augusta
Gela
Cagliari
Oristano
Porto Torres
Sardinia
Corsica
Sicily
Adriatic Sea
Tyrrhenian Sea
Ionian Sea
Mediterranean Sea
http://www.cia.gov/cia/publications/factbook/

Each day of the school is focussed on a different theme:
"principles & methods"
"joining things together"
"crystallographic implementations"
"selected topics in crystallography"
"special methods"

The City

Siena is described as one of the finest examples of a Medieval city. It is in the Italian province of Tuscany and has direct bus connection to Florence (1 hour) and Rome (3 hours).

The Venue

The Certosa di Pontignano has its origins as a medieval 14th century monastary. It is now run by the University of Siena. Attractively placed on the top of a hill, it is surrounded by vineyards; with a direct view to the town of Siena, and a famous Chianti winery.

School Aims

To have the crystallographic computing experts of the present, help train and inspire a generation of experts for the future. This will be achieved by the use of an excellent (and full) program of lectures, workshops and projects.

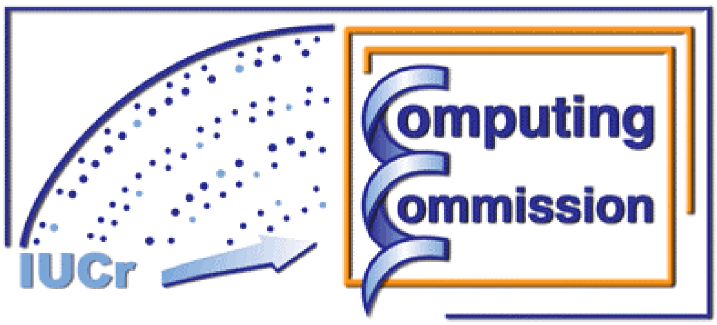
Università degli Studi di Siena

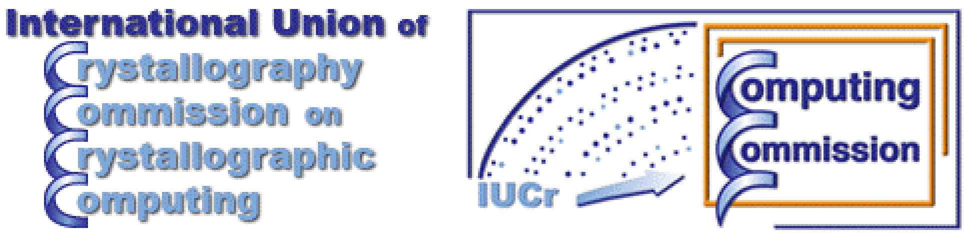
Winners of the CompComm Logo Competition

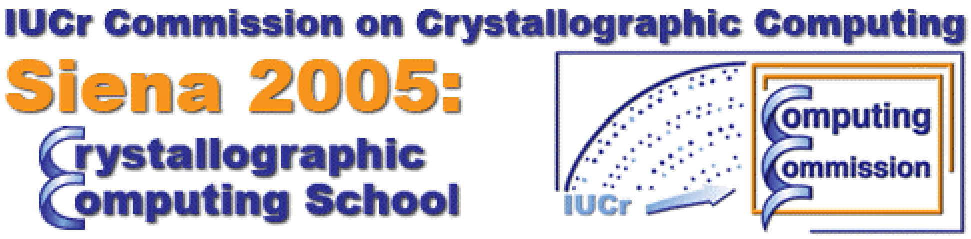
IUCr
Computing
Commission

Can people do better than this?

Indeed they can! With a greatful thankyou to the people who submitted their ideas, Paul (University of Florida, USA) and Kathy Sehnke were declared winners of the CompComm logo competition. Their stated synopsis was "This logo represents some important molecular structural components of which the computing commission works with and works towards. The X-ray film, diffraction pattern, computer, beta strand and helix are surrounded by a line which symbolizes the box that contained the Beevers-Lipson Strips. These elements within the logo were and are necessary for the end result of solving a molecular structure." The winning graphics are below: a) newsletter logo, b) webpage banner logo and c) Siena 2005 Computing School logo.







All submissions received are still viewable via:
http://www.iucr.org/iucr-top/comm/ccom/ccom/logo_sub.html

Refinement on weak or problematic small molecule data using SHELXL97

Alexander J. Blake,
School of Chemistry, The University of Nottingham, Nottingham UK
E-mail: A.J.Blake@Nottingham.ac.uk *- WWW:* http://www.nottingham.ac.uk/chemistry/staff/blake.html

Introduction

Problematic structure refinements can be the result of a number of factors, including poor diffraction, twinning, small crystal size, pseudosymmetry, significant absorption effects and positional disorder. (For a comprehensive review of difficult refinements see D.J. Watkin, *Acta Cryst.* 1994, A**50**, 411–437.) Such structures may exhibit low data/parameter ratios, apparent voids, extreme or unrealistic displacement parameters, high residual electron density or poorly defined hydrogen atom positions. While certain of these problems, such as absorption, are typically dealt with outside the refinement process, it will normally be necessary to incorporate a treatment for most of the others as part of refinement.

A principal feature of such refinements is the lack of information in the diffraction data pertaining to particular regions of the structure, for example in disordered regions where the disorder components are not resolved. In such cases the application of restraints (formally, the addition of extra observations) and/or constraints (formally, the fixing of certain parameters) can allow the refinement to proceed to a satisfactory conclusion. A simple example of a restraint derived from chemical knowledge would be one which states that the chemically-equivalent distances in the BF_4^- anion should be the same, while constraints would be appropriate to ensure that an atom on a special position did not wander away from it. The value of restraints and constraints depends crucially on their validity, with inappropriate ones distorting instead of assisting the refinement. Distance and other geometric restraints are sometimes familiar through previous experience of the same chemical moieties, while unfamiliar ones can be determined from database surveys or even from ordered examples on the same species within the structure being studied.

This article will focus primarily on a range of hexanuclear supramolecular cages constructed using a ligand (see Figure 1) which is both blocking and chelating. The structures were refined using *SHELXL*97 and recently published in *Chemical Communications* (O.V. Dolomanov, A.J. Blake, N.R. Champness, M. Schröder & C. Wilson, "A novel synthetic strategy for hexanuclear supramolecular architectures", *Chem. Commun*. 2003, pp. 682–683).

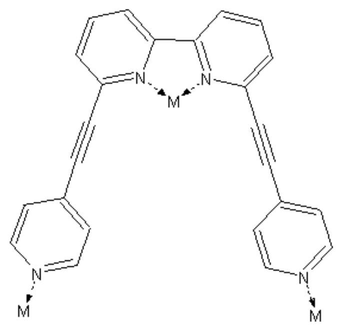
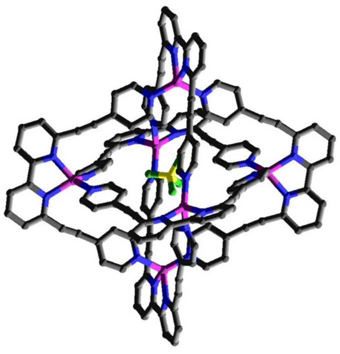
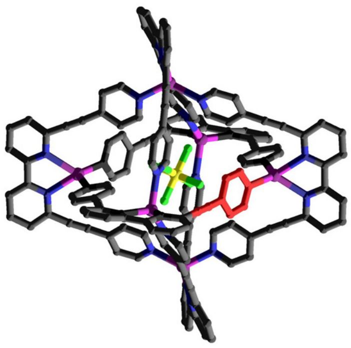
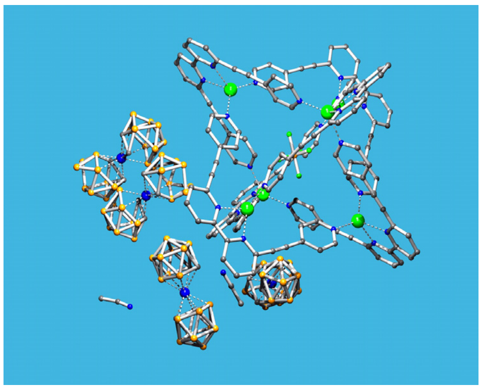
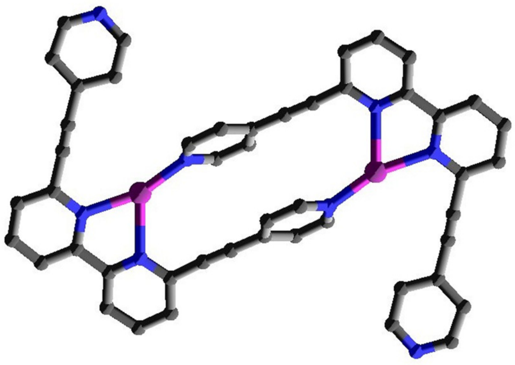


Figure 1: *Ligand which is both blocking and chelating.*

The structures are illustrated in Figures 2–5. The refinement problems include disorder in many anions, partial occupancy for several anions, low resolution data and low r/p ratios. Moreover, it is important to establish the identity and the locations of the anions, in order to understand how they contribute (*e.g.*, as templates) to the formation of the different structures.

Figures 2 and 3 : *hexanuclear cage {[$Cu_6L_6(BF_4)$](BF_4)$_5$}; hexanuclear cage {[$Ag_6L_6(SbF_6)$](SbF_6)$_5$}*

Figures 4 and 5 : *hexanuclear cage [$Ag_6L_6(BF_4)$][$Co(C_2H_{11}B_9)_2$]$_5$; dinuclear cation in (AgL)$_2$[$Co(C_2H_{11}B_9)_2$]$_2$*

Some of the refinement tools available in SHELXL97

(http://shelx.uni-ac.gwdg.de/SHELX/) (G.M. Sheldrick, University of Göttingen, Germany, 1997) [values in square brackets indicate defaults]

`EXYZ atomnames`
The same x, y and z parameters are used for all the named atoms.

`EADP atomnames`
The same isotropic or anisotropic displacement parameters are used for all the named atoms.

`PART n sof`
The following atoms belong to **PART n** of a disordered group with the site occupation factor (**sof**) shown.

`DFIX d s[0.02] atom pairs`
The distance pairs of atoms are restrained to a specified target value of **d** with standard uncertainty **s**.

`SADI s[0.02] atom pairs`
The distances between pairs of atoms are restrained to be equal with an effective s.u. of **s** (*cf.* **DFIX**)

`SAME s1[0.02] s2[0.02] atomnames`
The atoms specified here are linked to the same number of atoms which follow.

`FLAT s[0.1] four or more atoms`
The named atoms are restrained to lie in a plane.

`SUMP c sigma c1 m1 c2 m2 ...`
The linear restraint: c = c1*fv(m1) + c2*fv(m2) + ... is applied to the specified free variables.

`DELU/SIMU/ISOR`
Applies various restraints to anisotropic displacement parameters (not discussed here).

`FRAG code[17] a … γ`
Enables a fragment to be input using an input cell and coordinates.

`FEND`
This must immediately follow the last atom of a FRAG fragment.

`AFIX n>16`
Applies geometry of fragment with this **n** value.

Example 1: Simple distance restraints in a tetrahedral anion (BF_4^-)

The first **DFIX** line below applies a distance restraint to each of the four equivalent B–F bonded distances, the second restrains the shape to be a tetrahedron centered on the boron atom by restraining non-bonded F···F distances to be equal. The structure is shown in Figure 2 and the atom numbering in Figure 6.

```
DFIX   1.38  0.01    B F1    B F2    B F3    B F4
DFIX   2.25  0.02    F1 F2   F1 F3  F1 F4   F2 F3   F2 F4   F3 F4
```

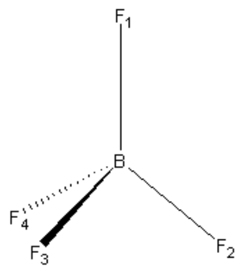


Figure 6 : *numbering of tetrahedral anion (BF_4^-)*

This procedure can serve two purposes, the first being to apply a sensible geometry to a single component. It can also serve another purpose: by defining a sensible geometry for one disorder component it can improve the definition of a second component. It would then be normal to apply the same pattern of restraints to this second (and any subsequent) disorder component. This situation is illustrated below, with the atom identification shown in Figure 7.

```
DFIX   1.38  0.01    B F1    B F2    B F3    B F4
DFIX   2.25  0.02    F1 F2   F1 F3   F1 F4   F2 F3   F2 F4   F3 F4

DFIX   1.38  0.01    B F1′  B F2′  B F3′  B F4′
DFIX   2.25  0.02    F1′ F2′  F1′ F3′   F1′ F4′   F2′ F3′   F2′ F4′   F3′ F4'
```

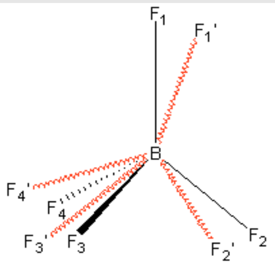


Figure 7 : *numbering of disordered tetrahedral anion (BF_4^-)*

It should then be possible to the refine the occupancy of F1 to F4 versus that of F1’ to F4’

Example 2: Geometric restraints for a less common anion

In this example (Figure 3) the anion (SbF_6^-) is less common and although we know it to be octahedral we are unsure of the Sb–F and F⋯F distances involved. In these circumstances we can exploit the geometric relationships among the parameters even although we are unsure of their actual values.

There is one Sb–F distance and two F⋯F distances corresponding to *cis* and *trans* dispositions of F.

```
SADI  0.01  Sb26  F27  Sb26 F28  … Sb26 F32
SADI  0.02  F27 F28   F27 F29  F27 F30  F27 F31  F27 F32  F28 F29 … F31 F32
SADI  0.02  F27 F30    F28 F31    F29 F32
```

Just as it is important to ensure that any explicit restraints (*e.g.*, distances specified using DFIX) are sensible, the outcome of the application of any similarity restraints should also be realistic. In this example the mean Sb–F distance was 1.86 Å, in very good agreement with the results of a subsequent search of the Cambridge Structural Database which yielded 175 entries with a mean value of 1.85(1) Å. We did not find any evidence of disorder but the restraints allowed us to refine a sensible geometry for light atoms (F, $Z = 9$) in the presence of a much heavier one (Sb, $Z = 51$) shown in Figure 8.

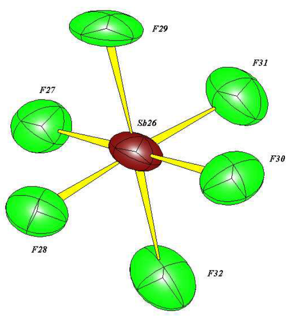


Figure 8 : *Refined SbF_6^- molecule*

Example 3: An ill-defined tripodal ligand

A schematic view of the ligand appears in Figure 9. The three arms emanating from the central nitrogen atom are chemically identical and would be expected to exhibit very similar bond distances and valence angles, but not necessarily the same torsion angles. Poor crystal quality led to a refinement which gave unrealistic bonds and angles and poor agreement between the geometry in different arms: this outcome simply looked unsatisfactory. The solution was to apply similarity restraints between each chain involving their bonded distances and valence angles, but no specific values were assigned to any parameter. Taking the lower chain (in black and numbered C1 to O7) in Figure 9 as the reference group, we first apply similarity restraints to the red chain on the left numbered C11 to O17.

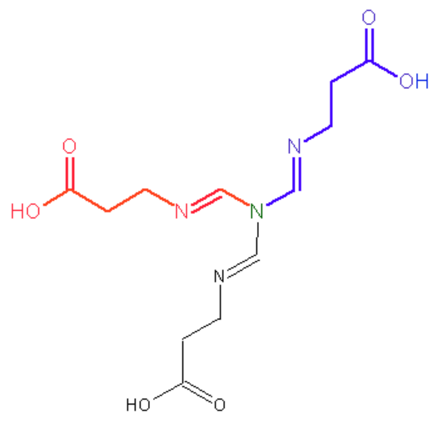


Figure 9 : *tripodal ligand.*

```
SAME  0.01  C1 > O7
C11   ...
N12 …
C13 …
C14 …
C15 …
O16 …
O17   ...
```

Then we do the same for the upper blue chain numbered C21 to O27.

```
SAME  0.01  C1 > O7
C21   ...
N22 …
C23 …
C24 …
C25 …
O26 …
O27   ...
```

The effect of this procedure is to average out the discrepancies between the three chains and achieve a better result for all three. As in the previous example, the final values for the geometric parameters form an important criterion for the validity of the procedure.

Example 4: A bromide anion disordered within a cavity

In the next example one bromide ion was required to balance the charge within a structure but instead of one large difference peak a number of difference peaks of moderate height were identified inside a cavity within the structure. The cavity was much larger than required to accommodate a bromide and there seemed to be no single preferred site for it. Five of the difference peaks stood out above the rest, so a model was developed based on these: this model is represented schematically by Figure 10.

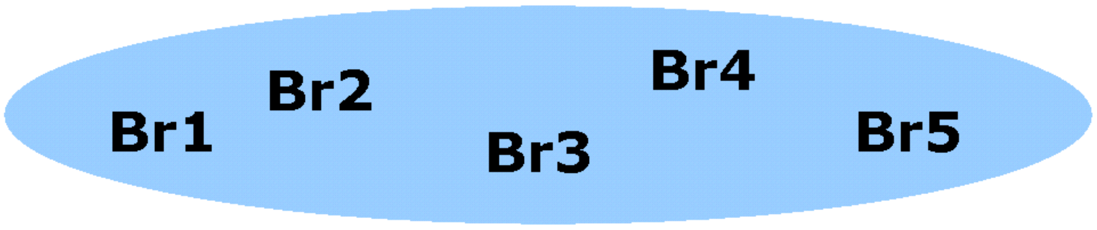


Figure 10 : *schematic diagram of bromide disordering in a cavity.*

Such a situation is likely to cause correlation effects if atom positions, occupancies and displacement parameters are refined completely freely: however, application of one simple restraint, namely that the sum of the occupancies must equal unity, was sufficient to overcome this problem. On each atom line (Br1 … Br5) there is a reference to a free variable which describes the occupancy of that atom; on the **FVAR** line, following the overall scale factor (**osf**) five free variables numbered 2 to 6 represent a starting occupancy of 0.2 for each bromine component; finally, on the **SUMP** line there appears the linear restraint that the sum of the individual occupancies should equal 1.00 with a standard uncertainty of 0.01.

```
SUMP  1.00   0.01   1  2   1  3   1  4   1  5   1  6

FVAR  osf   0.2  0.2  0.2  0.2  0.2

Br1      5   x y z   21
Br2      5   x y z   31
Br3      5   x y z   41
Br4      5   x y z   51
Br5      5   x y z   61
```

In refinement, the individual occupancies are allowed to vary but they are always subject to this restraint so that unrealistic values and correlation with displacement parameters can be avoided or minimised. The validity of the refinement can be judged in several ways, for example by the behaviour of the displacement parameters and by the standard uncertainties on the individual occupancies: in this case the criteria were all satisfactory for a model showing unequal occupancies of five sites. [With only two disorder components the analogous procedure is somewhat simpler, requiring no SUMP instruction and only a single free variable.]

Example 5: a large structure containing rigid $[Co(C_2B_9H_{11})_2]^-$ anions

This structure is shown in Figure 4. It is a big structure constructed using large anions, several of which occur in the asymmetric unit. When the structure was refined without restraints there were problems with the data/parameter ratio and with poor geometry for the anions. It would be possible, either manually or automatically, to generate multiple DFIX instructions for Co—B, Co—C, B—B, and B—C distances and to apply similarity restraints between the carbaborane cages: however, this would be complicated and might still allow the cages to distort. The rigidity of the anions allows a completely different approach to be adopted, one using constraints rather than restraints. A model for the anion can be taken from a variety of sources, including a more precise version in the same structure; a better version from another structure; a calculated or optimised version; or a typical or average database structure.

Having identified a suitable model, the first step is to import this model into the input file for the refinement program

```
FRAG    17   15.72  20.15 20.39 74.8 70.75 86.50
Co       4      x y z...
C1       1      x y z...
C2       1      x y z...
B3       3      x y z...
...
B19      3      x y z...
FEND
```

Note that the input model is bounded by the FRAG and FEND instructions and that the fragment has a number (17) which is greater than 16.

This model is then applied to the corresponding parts of the current structure, for example:

```
AFIX   17
Co1     7   0.33250   0.76245   0.52909  11.000   0.0608 0.1389 =
            0.0396   -0.0183   -0.0212    0.0153
C1      1   0.37668   0.84367   0.54903  11.00000    0.155
C2      1   0.41382   0.84350   0.45796  11.00000    0.089
B3      3   0.31680   0.82455   0.43612  11.00000    0.117
...
B19     3   0.20793   0.79538   0.51437  11.00000    0.138
AFIX    0
```

Before refinement proceeds this part is idealised to the input model; the AFIX 17 … AFIX 0 sequence is replaced by a simple rigid group refinement (AFIX 3); the FRAG ... FEND sequence of lines is no longer required and is not copied to the output model file. For each $[Co(C_2B_9H_{11})_2]^-$ anion the number of positional parameters is reduced from 69 to 6, the latter number corresponding to three positional and three orientational parameters for each rigid group. A similar approach was also successful for the structure shown in Figure 5.

This is a powerful and attractive technique but it can only be applied in certain circumstances. For example, the 3D matching requirements are stringent and the method cannot be applied to fragments where there is uncertainty or variability in the positions of any of the constituent atoms. As it is applied as a constraint, the input model must be completely valid and it is particularly important to check the program's refinement indicators for any warning of errors or problems.

Conclusion

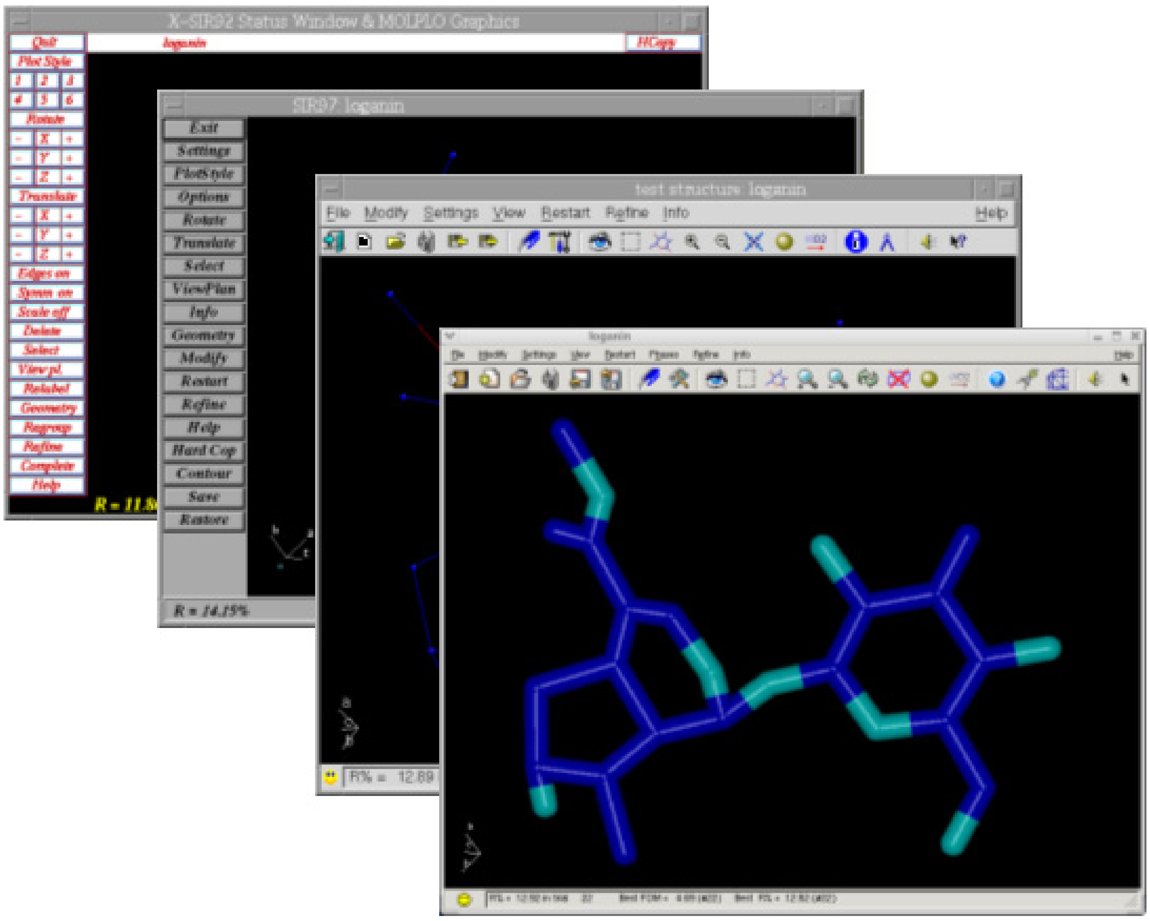
This article has shown how the careful choice and application of geometric restraints or constraints can allow acceptable refinement of problematic structures. As small molecule crystallography advances into larger and/or poorly diffracting structures these tools will be increasingly called upon to facilitate structure analysis.

Restraints and Constraints in Sir2004

Maria C. Burla[1], Rocco Caliandro[2], Mercedes Camalli[3], Benedetta Carrozzini[2], Giovanni L. Cascarano[2], Liberato De Caro[2], Carmelo Giacovazzo[2,4], Giampiero Polidori[1], Riccardo Spagna[3]
[1]Dipartimento di Scienze della Terra - Piazza Università, 06100 Perugia, Italy; [2]Istituto di Cristallografia, CNR, Via Amendola 122/o, 70125 Bari , Italy;[3]Istituto di Cristallografia, CNR, Sezione di Monterotondo, CP 10, Monterotondo Stazione, 00016 Roma, Italy; [4]Dipartimento Geomineralogico, Università di Bari, Campus Universitario, Via Orabona 4, 70125 Bari, Italy.
E-mail: riccardo.spagna@ic.cnr.it - WWW: http://www.ic.cnr.it/

Introduction

SIR (**S***emi-***I***nvariant* ***R****epresentation*) is a collection of Direct Methods programs for automatic crystal structure solution based on the *Representation Theory*[1,2]. The main elements of this family are Sir92[3], Sir97[4], Sir2002[5] and Sir2004 (in preparation).


X-SIR92 Status Window & MOLPLO Graphics
loganin
SIR97 loganin
test structure loganin
File Modify Settings View Restart Refine Info Help
loganin
R = 14.15%
R% = 12.89

All the programs are equipped with a graphical interface (GUI) to interact with the program. While in the older programs X11 libraries were used to build the GUI, in the newest version GTK graphic libraries have been used.

The main features of the Sir programs are:

- Minimal input information
- Possibility of complete automatism
- Reduced user intervention
- Simple (and powerful) graphic interface
- Run on almost all platforms

Sir92 was the first program, created by the Sir team and distributed to the scientific community, devoted to the automatic solution of crystal structures by direct methods.

In Sir97, it's evolution, powerful automatisms and new algorithms were added. Furthermore it was possible, *via* an improved graphical interface, the refinement and the analysis of the crystal structure model found by the program.

Both Sir92 and Sir97 were fast and powerful but their goal was small molecules.

Sir2002 constitutes the first approach of the Sir team to the solution of macromolecules: it is able to solve, without any user intervention, crystal structure up to 2000 atoms in the asymmetric unit, provided the data are at atomic resolution (better than 1.2Å). The strategy adopted in Sir2002 was to explore in sequence, via Direct Space Refinement and automatic Diagonal Least Squares, all the trials until a solution was found. This procedure, very powerful, was also time consuming particularly for large structures.

In Sir2004 the use of suitable Figures Of Merit[6] at an early stage of the phasing procedure allows to select the most promising trials. So potentially good solutions are separated from bad ones just after the Tangent Formula step. Only the most promising trials are submitted to Direct Space Refinement, so sparing computing time.

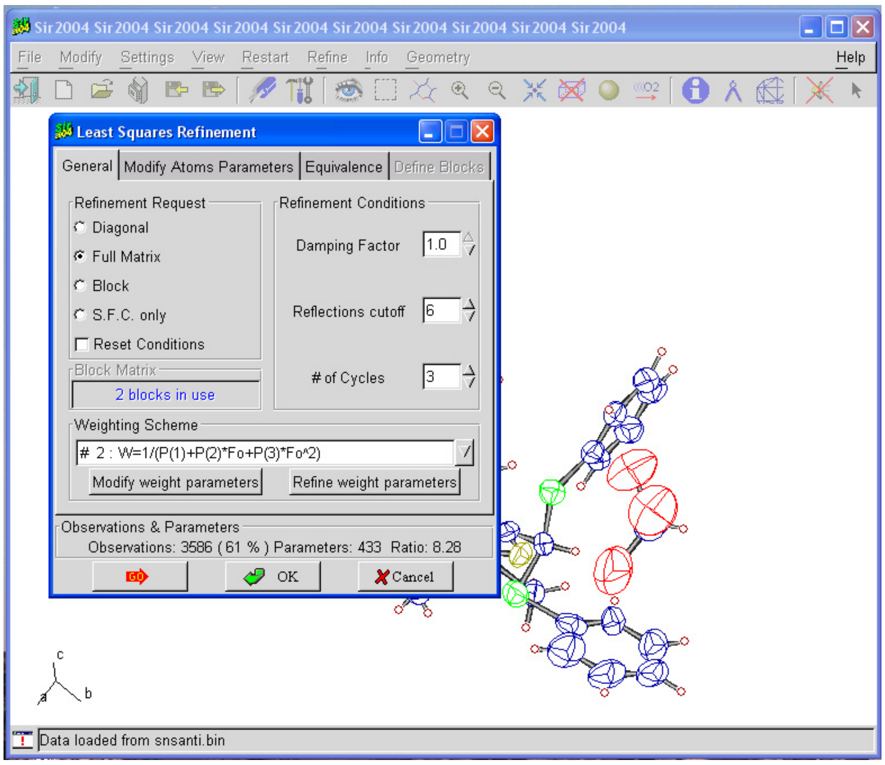
The 1.2Å limit on the atomic resolution of the data has been moved, in Sir2004, to 1.4-1.5Å[7,8]; new procedures able to attain and highlight the solution in these conditions have been developed.

Refinement of the model

The stage of the process of getting the best model of a crystal structure is to find the values of the atomic parameters which give the best fit between observed and calculated structure factors. This optimisation procedure is called refinement and we implemented in Sir2004 (provided that the resolution is sufficient) the well known and most generally used method of least squares.

The specific features of our refinement are the following.

1) The ability to reduce the full matrix of the normal equations defining any kind of blocks.
2) 18 weighting schemes are available. If the weighting scheme contains adjustable parameters, the program refines the values to obtain a good distribution of $<w\Delta^2>$ against $|F|$ and resolution and the value of the goodness of fit close to the unit.
3) The program generates constraints for the parameters of atoms on special positions in all space groups.


Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004
File Modify Settings View Restart Refine Info Geometry Help
Least Squares Refinement
General
Modify Atoms Parameters
Equivalence
Define Blocks
Modify weight parameters
Refine weight parameters
OK
Cancel
c
a
b

Refinement Request
- Diagonal
- [✓ Full Matrix]
- Block
- S.F.C. only
- Reset Conditions

Block Matrix: 2 blocks in use

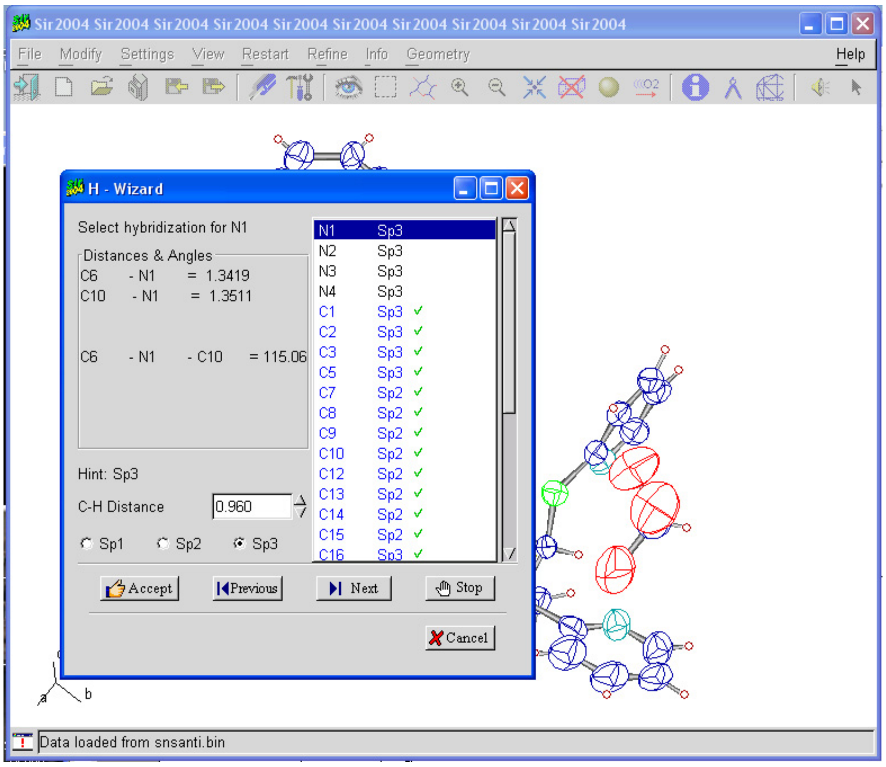
Refinement Conditions
- Damping Factor: 1.0
- Reflections cutoff: 6
- # of Cycles: 3

Weighting Scheme: # 2 : W=1/(P(1)+P(2)*Fo+P(3)*Fo^2)

Observations & Parameters
Observations: 3586 (61 %) Parameters: 433 Ratio: 8.28

Data loaded from snsanti.bin

4) Automatic or through wizard generation of hydrogen atoms and their contributions are included in the refinement allowing the positional parameters to ride on the corresponding parent atom.


Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004
File Modify Settings View Restart Refine Info Geometry Help
H - Wizard
Accept
Previous
Next
Stop
Cancel
a
b

Select hybridization for N1

Distances & Angles
```
C6    - N1    = 1.3419
C10   - N1    = 1.3511

C6    - N1    - C10    = 115.06
```

Hint: Sp3

C-H Distance: 0.960

Sp1 Sp2 [✓ Sp3]

N1	Sp3	
N2	Sp3	
N3	Sp3	
N4	Sp3	
C1	Sp3	✓
C2	Sp3	✓
C3	Sp3	✓
C5	Sp3	✓
C7	Sp2	✓
C8	Sp2	✓
C9	Sp2	✓
C10	Sp2	✓
C12	Sp2	✓
C13	Sp2	✓
C14	Sp2	✓
C15	Sp2	✓
C16	Sp3	✓

Data loaded from snsanti.bin

5) The possibility to impose conditions (constraints) or additional information (restraints).
6) Floating origin is restrained automatically by setting the restrain of the sum of the. appropriate coordinate
7) Refinement of the Flack parameter to evaluate the absolute configuration.

Restraints

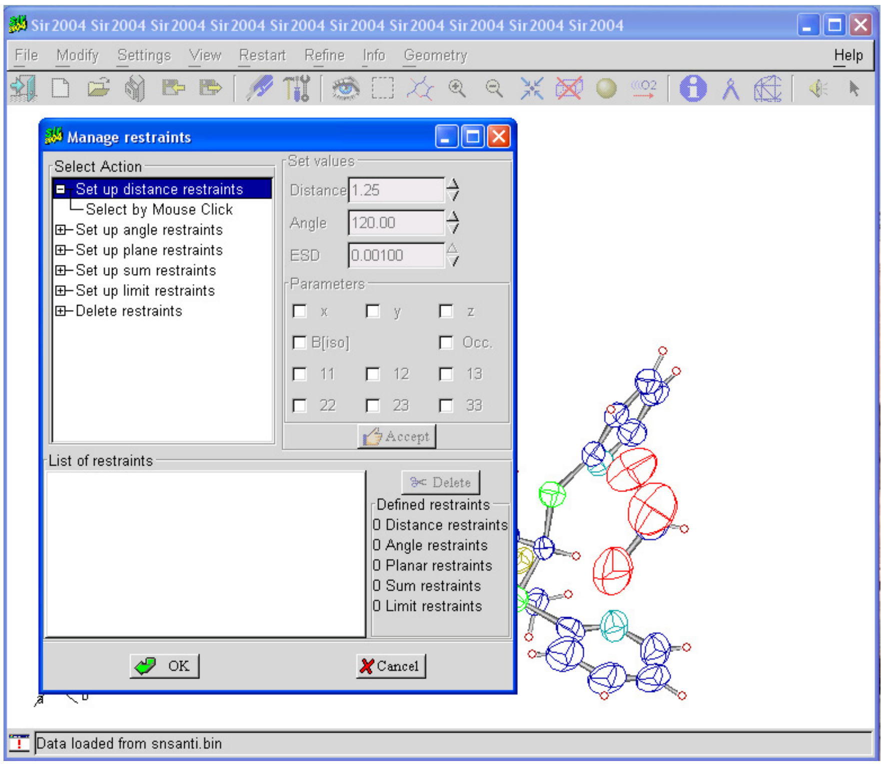
With poor data, a simple way to improve the convergence of the atomic parameters is to incorporate constraints and/or restraints in the refinement. A restraint is a function of the coordinates that is lowest when the coordinates are "ideal", and which increases as the coordinates become less ideal. Bond lengths, angles, etc. are "measurements" that must be fit by the model and can be considered as supplementary observations. The minimization function becomes:

$$M = \sum_{\mathrm{H}} w_{\mathrm{H}} \left(|F_{\mathrm{H}}|_{obs} - |F_{\mathrm{H}}|_{calc} \right)^2 + \sum_{q} w_q \left(g_{obs} - g_{ideal} \right)^2$$

where g_{obs} is a function of the coordinates and g_{ideal} is his ideal value, w_q is the $1/\sigma^2$ multiplied by the square of the of the goodness of fit for the reflection data.

The following restraints are available.

1) Geometrical conditions: bond distances, bond angles, planarity. Clicking on the item in the "Manage restraints" window the user chooses the requested restraints and then he has to select the atoms involved by mouse click. The planarity of a group of atoms is obtained by restraining their distances from the least squares plane of the group.


Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004
File Modify Settings View Restart Refine Info Geometry Help
Manage restraints

Select Action
- Set up distance restraints
  - Select by Mouse Click
- Set up angle restraints
- Set up plane restraints
- Set up sum restraints
- Set up limit restraints
- Delete restraints

Set values
Distance 1.25
Angle 120.00
ESD 0.00100

Parameters
x y z
B[iso] Occ.
11 12 13
22 23 33

Accept

List of restraints

Delete

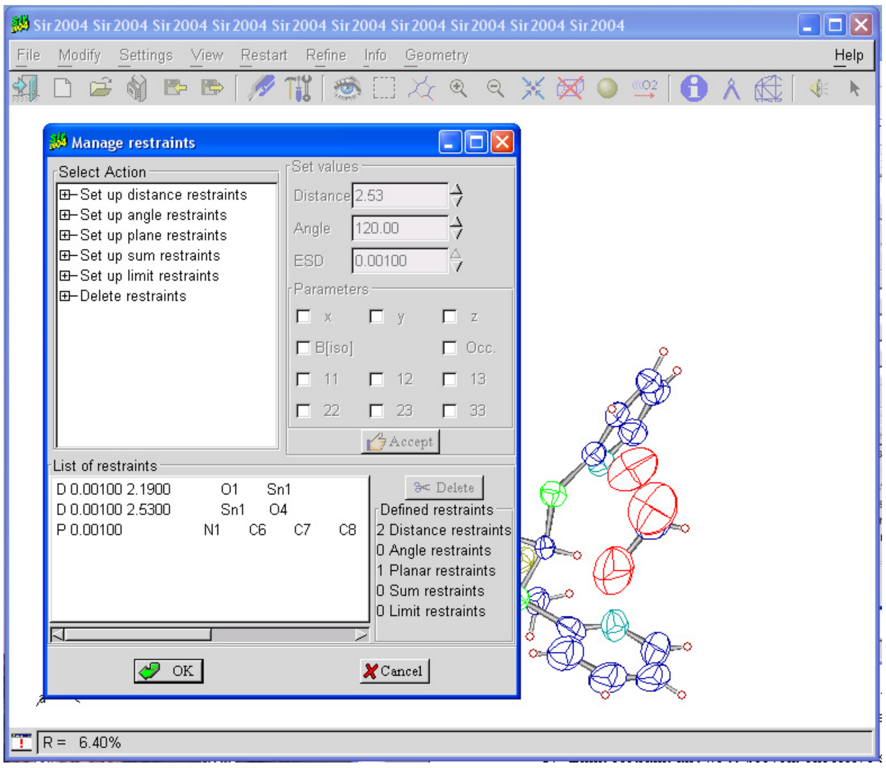
Defined restraints
0 Distance restraints
0 Angle restraints
0 Planar restraints
0 Sum restraints
0 Limit restraints

OK Cancel

Data loaded from snsanti.bin

2) The restraint of the sum of selected parameters could be useful in case of disorder, as well as in the non centrosymmetric space groups with floating origin, where this restraints are generated automatically by the program.
3) Limit restraint allows to prevent excessive shifts for some ill-conditioned atomic parameters.

The list of defined restraints are shown in the proper window and the user can delete or add new restraints at any moment of the process of refinement.


Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004 Sir2004
File Modify Settings View Restart Refine Info Geometry Help
Manage restraints

Select Action
- Set up distance restraints
- Set up angle restraints
- Set up plane restraints
- Set up sum restraints
- Set up limit restraints
- Delete restraints

Set values

Distance 2.53

Angle 120.00

ESD 0.00100

Parameters: x, y, z, B[iso], Occ., 11, 12, 13, 22, 23, 33

Accept

List of restraints

```
D 0.00100 2.1900        O1    Sn1
D 0.00100 2.5300        Sn1   O4
P 0.00100               N1    C6    C7    C8
```

Delete

Defined restraints
- 2 Distance restraints
- 0 Angle restraints
- 1 Planar restraints
- 0 Sum restraints
- 0 Limit restraints

OK Cancel

R = 6.40%

Constraints

A constraint is a mathematical relationship which imposes predefined conditions between atoms of the crystal structure or part of it. This technique has the advantage to reduce the number of parameters to refine, particularly useful when the ratio between atomic parameters and measured intensities is low. The atoms involved have to be regularized to a ideal model structure of known geometry (for example, benzene ring) and this rigid body is refined as compact unit assuming three translational parameters and three angles which define its orientation.

The method used to compute the coordinates of the model follows the approach described by Arnott & Wonacott (1966). In order to build the internal Cartesian coordinates, the program uses the ASCII file Sir2004.gru which contains models described using the Z-matrix format. The user can add models following the rules specified in the web site:
http://www.cineca.it/manuali/Unichem/5500/5500_248.html

Example of the file Sir2004.gru

```
Rigid Groups file 26 June 2003
Z-matrix
benze   6
    1   0          0 0       0 0    0
    2   1    1.455 0         0 0    0
    3   2    1.455 1 120.0 0        0
    4   3    1.455 2 120.0 1 0.0
    5   4    1.455 3 120.0 2 0.0
    6   5    1.455 4 120.0 3 0.0
benzene
```

The user has to indicate the label of the model contained in the Sir2004.gru file (**benz** in the example) and the corresponding labels of the atoms. In the current version, the graphic interface is in progress and the program get this information via input file.

Example of input file

```
%group
benz 1 6
 c10 c11 c12 c13 c14 c15
benz 2 6
 c16 c17 c18 c19 c20 c21
benz 3 6
 c23 c24 c25 c26 c27 c28
end
```

Sir2004 reads the following input file to execute the refinement applying these constraints:

```
%lsq
full
group  benz1   benz2  benz3 x y z theta phi psi
end
%end
```

References

1 - Giacovazzo, C. (1977). *Acta Cryst.*, A**33**, 933-944.
2 - Giacovazzo, C. (1980). *Acta Cryst.*, A**36**, 362-372.
3 - Altomare, A., Cascarano, G., Giacovazzo, C., Guagliardi, A., Burla, M.C., Polidori, G. & Camalli, M. (1994)**.** *J. Appl. Cryst.* **27**,435.
4 - Altomare, A., Burla, M.C., Camalli, M., Cascarano, G., Giacovazzo, C., Guagliardi, A., Moliterni, A.G.G., Polidori, G. & Spagna, R. (1999). *J. Appl. Cryst.*, **32**, 115-119.
5 - Burla, M.C., Camalli, M., Carrozzini, B., Cascarano, G.L., Giacovazzo, C., Polidori, G. & Spagna, R. (2003). *J. Appl. Cryst.*, **36**, 1103.
6 - Burla, M.C., Caliandro, R., Carrozzini, B., Cascarano, G.L., De Caro, L., Giacovazzo, C. & Polidori, G. (2004). *J. Appl. Cryst.,* **37**, in press**.**
7 - Burla, M.C., Carrozzini, B., Cascarano, G.L., De Caro, L., Giacovazzo, C. & Polidori, G. (2003). *Acta Cryst.* A**59**, 245-249.
8 - Burla, M.C., Carrozzini, Caliandro, R., Cascarano, G.L., De Caro, L., Giacovazzo, C. & Polidori, G. (2003). *Acta Cryst.* A**59**, 560-568

cctbx news: Geometry restraints and other new features

Ralf W. Grosse-Kunstleve, Pavel V. Afonine and Paul D. Adams,
Computational Crystallography Initiative, Lawrence Berkeley National Laboratory, One Cyclotron Road, BLDG 4R0230, Berkeley, California, 94720-8235, USA - Email : RWGrosse-Kunstleve@lbl.gov ; WWW: http://cci.lbl.gov/

1: Introduction

The Computational Crystallography Toolbox (cctbx, http://cctbx.sourceforge.net/) is the open-source component of the Phenix project (http://www.phenix-online.org/). Currently we are finalizing an initial version of the Phenix refinement procedures. The emphasis of this article is an introduction to the underlying open-source libraries for the handling of geometry restraints, molecular mask calculations, bulk-solvent correction, likelihood-based target functions for crystallographic refinement, and the relative scaling between these target functions and the geometry restraints.

Some of the functionality covered in this newsletter is implemented in the new top-level `mmtbx` module ("macro-molecular toolbox") of the cctbx project. Due to technical reasons the `mmtbx` source code is not currently hosted at the SourceForge site even though it is covered by the same open license as the rest of the cctbx project. However, the full `mmtbx` sources are included in the bundles available at the http://cci.lbl.gov/cctbx_build/ download site. For the future we are planning to move the `mmtbx` code to the SourceForge site.

In order to interactively run the examples scripts shown below, the reader is highly encouraged to visit http://cci.lbl.gov/cctbx_build/ and to download one of the completely self-contained, self-extracting binary cctbx distributions (supported platforms include Linux, Mac OS X, Windows, IRIX, and Tru64 Unix). On recent machines the installation requires significantly less than one minute of time. Even on the slowest machine available to us (SGI O2, R5000, 300MHz) a binary installation takes less than three minutes without requiring any manual intervention. A cctbx installation is non-intrusive and does *not* require system privileges. Traceless removal is as easy as running `rm -rf` or dragging a single folder to the Recycle Bin. Nobody will know you did it!

All example scripts shown below were tested with cctbx build 2004_08_05_0113.

2: from cctbx import geometry_restraints

Commonly refinement programs support inclusion of prior chemical knowledge such as bond lengths and bond angles via *geometry restraints*. The cctbx implementation of six types of geometry restraints is located in the `cctbx.geometry_restraints` module. The restraint types available are:

- bond
- nonbonded repulsion
- angle
- dihedral (same as torsion)
- chirality
- planarity

The `cctbx.geometry_restraints` module is designed as a uniform library to support both small-molecule and macro-molecular refinement. In general the requirements for small-molecule and macro-molecular refinement are quite different. For example, some macro-molecular refinement programs have limited or no support for symmetry bonds (i.e. bonds to atoms generated by symmetry), or bonds involving sites on special positions. This is not surprising because of the 27141 `pdb*.ent` files found at `ftp.rcsb.org` on July 27, 2004, only 256 include `LINK` records defining symmetry bonds (out of a total of 7078 files with `LINK` records), and only 534 files include heavy atoms on special positions (out of a total of 2746 files with atoms on special positions; most are water molecules). This means a little less than 98% of all macro-molecular structures can be refined with a program that does not correctly handle symmetry bonds or special positions.

In contrast, special positions and symmetry bonds are the norm in small-molecule crystallography, not the exception. The is particularly true for inorganic materials. In theory, a system that handles geometry restraints for the refinement of small molecules and inorganic materials will therefore immediately be able to handle all symmetry aspects of 100% of all macro-molecular structures. In practice however many small-molecule programs do not lend themselves to be used for macro-molecular work. This is due to fairly trivial nuisances such as unsuitable compiled-in limits on the number of atoms or bonds that can be processed, or more seriously, use of algorithms that scale with the square of the number of atoms and become prohibitively slow for large macro-molecular structures. Even more seriously, aspects that are crucial for the handling of macro-molecular structures may not be covered at all, such as nonbonded interactions, dihedral or chirality restraints.

The `cctbx.geometry_restraints` module was designed under the "completeness and correctness first, optimize later" paradigm. The handling of all symmetry aspects of bonded and nonbonded pair interactions is as complete as one expects to find in a small-molecule application, but the algorithms and data structures are optimized for handling a large number of atoms. I.e. the macro-molecular field will benefit from the rigorous treatment of symmetry, and the small-molecule field will benefit from speed increases. Owing to the facilities provided by the modern programming languages used (Python and C++), compiled-in limits are a problem of the past. The memory for all data is dynamically allocated.

2.1: cctbx.geometry_restraints.bond

Given the Cartesian coordinates of two bonded sites, the ideal bond length, and a weight, we can run the following Python code:

```
from cctbx import geometry_restraints
bond = geometry_restraints.bond(
  sites=[(1,2,3),(2,3,4)],
  distance_ideal=2,
  weight=10)
print "distance_model:", bond.distance_model
print "delta:", bond.delta
print "residual:", bond.residual()
print "gradients:", bond.gradients()
```

Output:

```
distance_model: 1.73205080757
delta: 0.267949192431
residual: 0.717967697245
gradients: ((3.0940107675850306, 3.0940107675850306, 3.0940107675850306),
(-3.0940107675850306, -3.0940107675850306, -3.0940107675850306))
```

The `bond` class performs all the basic computations required for gradient-driven refinement. The `residual()` is the contribution of this bond to the total "energy" of the geometry term of the target function and defined in the usual way (e.g. Hendrickson, 1985) as `weight * bond.delta**2`, where `bond.delta` is the difference `bond.distance_ideal - bond.distance_model`.

2.2: cctbx.geometry_restraints.bond_simple_proxy

Of course, during structure refinement the coordinates are changed. Therefore we need a data structure, in new speak an object, with some type of reference to the bonded sites along with `distance_ideal` and `weight`. We call this object `bond_simple_proxy` and it works like this:

```
from cctbx import geometry_restraints
from cctbx.array_family import flex
sites_cart = flex.vec3_double([
  (1,2,3),
  (2,3,4),
  (1,3,5)])
bond_proxy_1 = geometry_restraints.bond_simple_proxy(
  i_seqs=[0,1],
  distance_ideal=2,
  weight=10)
bond_proxy_2 = geometry_restraints.bond_simple_proxy(
  i_seqs=[1,2],
  distance_ideal=1.8,
  weight=20)
for bond_proxy in [bond_proxy_1, bond_proxy_2]:
  bond = geometry_restraints.bond(
    sites_cart=sites_cart,
    proxy=bond_proxy)
  print "sites:", bond.sites
  print "residual:", bond.residual()
```

Output:

```
sites: ((1.0, 2.0, 3.0), (2.0, 3.0, 4.0))
residual: 0.717967697245
sites: ((2.0, 3.0, 4.0), (1.0, 3.0, 5.0))
residual: 2.97662350914
```

`sites_cart` is an array of Cartesian coordinates for three sites. The `i_seq` (Index into SEQuence of sites) are the references mentioned above; they are simply integer indices into the `sites_cart` array. A `bond_simple_proxy` is essentially a `bond` with one level of indirection. We can turn a `bond_simple_proxy` into a `bond` by providing the `sites_cart` array referenced to by the `i_seq`. Then we can use the methods of the `bond` object to obtain the desired information as shown before, for example the `residual()` as in this example or the `gradients()` as in the previous example.

2.3: cctbx.geometry_restraints.shared_bond_simple_proxy

In all likelihood we will have to handle a considerable number of bonds. Therefore the next data structure we need is an array of bond proxies. The previous example can be rewritten to use "shared" arrays with bond proxy objects as the elements:

```
bond_proxies = geometry_restraints.shared_bond_simple_proxy()
bond_proxies.append(geometry_restraints.bond_simple_proxy(
  i_seqs=[0,1],
  distance_ideal=2,
  weight=10))
bond_proxies.append(geometry_restraints.bond_simple_proxy(
  i_seqs=[1,2],
  distance_ideal=1.8,
  weight=20))
for bond_proxy in bond_proxies:
  bond = geometry_restraints.bond(
    sites_cart=sites_cart,
    proxy=bond_proxy)
```

If the number of bonds is very large as in the case of macro-molecular structures, the Python `for` loop will become a performance bottleneck. Python is a dynamically typed language and therefore very convenient to use, but the convenience is payed for with a runtime penalty of one to two orders of magnitude. The remedy is to reimplement the Python loop in C++ and to do the *vector operation* at the speed of a compiled language. Using Boost.Python (http://www.boost.org/libs/python/doc/, see also Grosse-Kunstleve & Adams, 2003), it is easy to make the C++ function available in Python. In this way we can, for example, obtain all `bond.delta` values with a single call from Python to C++:

```
bond_deltas = geometry_restraints.bond_deltas(
  sites_cart=sites_cart,
  proxies=bond_proxies)
print list(bond_deltas)
```

Output:

```
[0.2679491924311227, 0.38578643762690501]
```

The idea behind this approach is similar to the idea behind vector computers. Python is the slow but general scalar unit, C++ the fast but restricted vector unit. Filling the array of bond proxies is similar to loading the vector unit and the call from Python to C++ is the vector operation. More on the subject of combining Python and C++ can by found in the Newsletter No. 1 in this series (Grosse-Kunstleve & Adams, 2003).

As an aside, the array of bond proxies is called a "shared" array because it may have multiple owners. The lifetime of shared arrays is controlled by a reference count. If the reference count goes to zero (because all owning references go out of scope or are deleted explicitly) the memory for the array is automatically deallocated. This is one of the fundamental mechanisms used by Python and C++ for making memory management simple (compared to FORTRAN) and at the same time safe (compared to C).

3: Symmetry: Friend or Foe?

The astute reader will have noticed that symmetry was not mentioned in the introduction to the `bond`, `bond_simple_proxy`, and `shared_bond_simple_proxy` objects. How does the symmetry come into play? The `simple` part of the two latter symbols is already a hint that there must be something more complex, and that is of course the symmetry. While symmetry is always nice to look at and therefore appears to be a friend, when it comes to writing algorithms for the handling of symmetry it quickly becomes apparent that symmetry is a pretty bad foe. Symmetry introduces singularities and each singularity requires special attention. For example, the 230 crystallographic space groups can be understood as 230 unique singularities, each of which has a different set of singular positions known as special positions. Needless to say, each singularity requires a name or number and therefore we have space group symbols and numbers, Wyckoff tables, Wyckoff letters, site symmetry symbols, etc., etc. As a rule of thumb, the source code required for handling a problem complete with symmetry is at least ten times the amount of source code required for the "simple" case. The handling of symmetry pair interactions and pair interactions involving sites in special positions is, unfortunately, not an exception.

We use the term *pair interaction* with reference to both bonded and nonbonded interactions. It comes as a little relief that the handling of bonded and nonbonded pair interactions is very similar. The main difference is the function used to compute the contributions to the total energy term for the geometry. In the case of bonded interactions it is the simple harmonic function `weight * bond.delta**2`, in the case of nonbonded interactions it is a more involved function of exponentials. However, up to the point of determining the `distance_model` required in both cases the algorithms are identical.

3.1: cctbx.crystal.direct_space_asu.asu_mappings

One important term we forgot to mention in the list of names and numbers required for the singularities introduced by symmetry is that of *asymmetric unit*. Before we can introduce symmetry pair interactions we have to get acquainted with the `cctbx.crystal.direct_space_asu.asu_mappings` class. The development of this class is based on the work published in the Newsletter No. 2 in this series (Grosse-Kunstleve et al., 2003). The web pages at http://cci.lbl.gov/asu_gallery/ are available for viewing the shapes of the standard asymmetric units as defined in the International Tables for Crystallography, Volume A. These shapes play a fundamental role in all cctbx algorithms involving pair interactions.
Pair interactions are commonly considered up to a certain cutoff distance, for example a maximum bond length when searching for bonds, or a maximum nonbonded distance when searching for nonbonded interactions. A fundamental consideration is that all pair interactions can be mapped by symmetry into the shape of the standard asymmetric unit expanded by a buffer region equivalent to the chosen cutoff distance. Let's dig out the simple `quartz_structure` introduced in the Newsletter No. 1 to see how this works in practice:

```
from cctbx import xray
from cctbx import crystal
from cctbx.array_family import flex
quartz_structure = xray.structure(
  crystal_symmetry=crystal.symmetry(
    unit_cell=(5.01,5.01,5.47,90,90,120),
    space_group_symbol="P6222"),
  scatterers=flex.xray_scatterer([
    xray.scatterer(
      label="Si",
      site=(1/2.,1/2.,1/3.),
      u=0.2),
    xray.scatterer(
      label="O",
      site=(0.197,-0.197,0.83333),
      u=0)]))
quartz_structure.show_summary().show_scatterers()
asu_mappings = quartz_structure.asu_mappings(buffer_thickness=2)
print "n_sites_in_asu_and_buffer:", asu_mappings.n_sites_in_asu_and_buffer()
```

The second to last line is a high-level interface provided by the `xray.structure` class for performing the process mentioned in the previous paragraph. First, the standard asymmetric unit is determined via lookup in the "reference file" introduced in Newsletter No. 2 and, if necessary, a change-of-basis transformation from the reference setting of the space group symmetry to the given setting (in the example the given setting is already the reference setting). Next, the asymmetric unit is expanded by moving the facets 2 Å out to generate the buffer region. Finally the space group symmetry is applied to the sites in order to fill the asymmetric unit including the buffer region. The end result is an instance of the class `cctbx.crystal.direct_space_asu.asu_mappings`. The output of running the example is:

```
Number of scatterers: 2
At special positions: 2
Unit cell: (5.01, 5.01, 5.47, 90, 90, 120)
Space group: P 62 2 2 (No. 180)
Label, Scattering, Multiplicity, Coordinates, Occupancy, Uiso
Si   Si      3 ( 0.5000  0.5000  0.3333) 1.00 0.2000
O    O       6 ( 0.1970 -0.1970  0.8333) 1.00 0.0000
n_sites_in_asu_and_buffer: 20
```

To understand the workings of the `asu_mappings` object we start with the `asu_mappings.mappings()` array provided by the object. For each scatterer in our `quartz_structure` there is one entry in the mappings array:

```
assert asu_mappings.mappings().size() == quartz_structure.scatterers().size()
for mappings in asu_mappings.mappings():
  print type(mappings), len(mappings)
```

Output:

```
<type 'tuple'> 3
<type 'tuple'> 17
```

This tells us that each element of the `asu_mappings.mappings()` array is a standard Python tuple, i.e. a list-like sequence of Python objects. We also learn that the first tuple has 3 elements and the second tuple has 17 elements. Each element represents exactly one site in the asymmetric unit or the buffer region. The first element of each tuple is always for the site in the asymmetric unit; by definition there can only be one. All following elements of each tuple represent sites in the buffer region. I.e. in this case there are 2 Si atoms in the 2 Å buffer region and 16 O atoms. To find out where they are we can use other facilities provided by the `asu_mappings` object. To keep the output short we concentrate on the 3 mappings for the Si atom:

```
for mapping in asu_mappings.mappings()[0]:
  print "i_sym_op:", mapping.i_sym_op()
  print "unit_shifts:", mapping.unit_shifts()
  print "symmetry operation:", asu_mappings.get_rt_mx(mapping)
  print
```

Output:

```
i_sym_op: 2
unit_shifts: (0, 0, -1)
symmetry operation: y,-x+y,z-1/3

i_sym_op: 0
unit_shifts: (0, 0, 0)
symmetry operation: x,y,z

i_sym_op: 1
unit_shifts: (1, 0, -1)
symmetry operation: x-y+1,x,z-2/3
```

Each `mapping` object stores the number of the symmetry operation and the unit shifts that were used to map the original site to the site in the asymmetric unit or the buffer region. To enforce consistency, a complete copy of the symmetry operations is stored inside the `asu_mappings` object. This is enables us to use `asu_mappings.get_rt_mx(mapping)` to compute the final symmetry operations giving the `mapping` object. ("`rt_mx`" stands for "rotation-translation matrix").

3.2: cctbx.crystal.neighbors_fast_pair_generator

We know that the silicon atoms in the `quartz_structure` are covalently connected to the oxygen atoms and that the Si-O bond distance is around 1.6 Å. This is how we find the bonds:

```
pair_generator = crystal.neighbors_simple_pair_generator(
  asu_mappings,
  distance_cutoff=1.7)
for pair in pair_generator:
  print pair.i_seq, pair.j_seq, pair.j_sym, pair.dist_sq**.5
```

Output:

```
0 1 0 1.61598604691
0 1 1 1.61598604691
0 1 12 1.61598604691
0 1 15 1.61598604691
1 0 1 1.61598604691
```

The `pair_generator` is a Python iterator that performs a simple-minded search with approximately N*N/2 iterations for all pair interactions within the given `distance_cutoff` of 1.7 Å, where N is the number of atoms. At each iteration we obtain a `pair` object with integer references into the `asu_mappings.mappings()` array as introduced in the previous section. The indices `pair.i_seq` and `pair.j_seq` are indices into the `asu_mappings.mappings()` array. The index `pair.j_sym` is an index into the `asu_mappings.mappings()[pair.j_seq]` tuple (see previous section). To avoid redundancies, only bonds that emanate from within the asymmetric unit are considered. Therefore we do not need a corresponding `i_sym` index; it is always 0. I.e. the three integer indices are sufficient to uniquely define a bond based on the `asu_mappings` object.

Alternatively we could generate the same list of pairs with the "fast" pair generator:

```
pair_generator = crystal.neighbors_fast_pair_generator(
  asu_mappings,
  distance_cutoff=1.7)
```

This alternative pair generator is designed for structures with a large number of sites. The interfaces of the simple and the fast pair generators are identical, but internally the fast generator is much more complex. The asymmetric unit including the buffer region is subdivided into cubes with a vertex length equivalent to `distance_cutoff`. In the search for pair interactions involving a given pivot site, only the cube of the pivot site and the 26 surrounding cubes have to be considered. The average number of sites per cube is approximately independent of the size of the structure. For a large number of sites the search time will therefore scale approximately linearly with the number of cubes instead of quadratically with the number of sites. This leads to dramatic increases in speed. For example (Linux, Xeon 2.8GHz):

```
number of atoms    time simple search    time fast search
  3500 (gere)           1.6 seconds           0.1
 59000 (groel)        377.4                   1.5
```

In practice there is no good reason for using the simple version of the pair generator. The main reason for keeping it in the library is to support a regression test that validates the fast generator.

3.3: cctbx.crystal.pair_asu_table

The `cctbx.crystal.pair_asu_table` is the center piece of the cctbx system for the handling of pair interactions involving symmetry. The internal `process_pair` member function of this C++ extension class is the heart of the center piece. It is responsible for generating symmetrically equivalent pair interactions and for the removal of redundant interactions. A full description of the algorithm implemented by the `process_pair` function is beyond the scope of this article even though the C++ source code comprises only 41 lines (see file `cctbx/include/cctbx/crystal/pair_tables.h`). However, the following example demonstrates the most important features:

```
pair_asu_table = crystal.pair_asu_table(asu_mappings=asu_mappings)
pair_asu_table.add_all_pairs(distance_cutoff=1.7)
```

The `pair_asu_table.add_all_pairs(distance_cutoff=1.7)` statement uses the fast pair generator as described in the previous section. When the first pair is processed by the `process_pair` function, the site symmetries of the two sites involved are applied to generate all symmetrically equivalent pairs. For the simple quartz structure, this step will already generate all pairs and add them to the `pair_asu_table` object. The pairs subsequently produced by the pair generator are found by lookup in the internal table and no further processing is necessary. At this stage the `pair_asu_table.table()` object managed by the `pair_asu_table` object will hold the data:

```
pair_asu_table.show()
```

Output:

```
i_seq: 0
  j_seq: 1
    j_syms: [0, 1, 12, 15]
i_seq: 1
  j_seq: 0
    j_syms: [0, 1]
```

`pair_asu_table.table()` is the most deeply nested data structure in the entire cctbx. In Python terms it is a *list of dictionaries associating integers with lists of lists*. If this appears overly complicated consider Einstein's famous quote: "Make everything as simple as possible, but not simpler." We are certain that `pair_asu_table.table()` is as simple as possible because each level of nesting represents a clear concept necessary to fully characterize symmetry pair interactions:

- The outermost list holds one entry per atom. The `i_seq` index is implied.
- Each entry is a dictionary. The keys are the `j_seq` indices.
- The value corresponding to each `j_seq` index is a list of groups of `j_sym` indices.
- The interactions defined by the `j_sym` indices in each group are symmetrically equivalent.

Since the interactions are fully characterized it is now very simple to extract the interactions unique under symmetry. Since we are not concerned about the directionality of the pair interactions (i.e. A-B is the same as B-A) we only have to consider interactions with `i_seq <= j_seq`, and we only need the first element from each group of symmetrically equivalent interactions. This procedure is implemented as the `extract_pair_sym_table` method of `pair_asu_table`:

```
pair_sym_table = pair_asu_table.extract_pair_sym_table()
pair_sym_table.show()
```

Output:

```
i_seq: 0
  j_seq: 1
    -y,x-y,z-1/3
i_seq: 1
```

This shows that the quartz structure has only one unique Si-O bond under symmetry.
An important point to note is that `pair_sym_table` is, in contrast to `pair_asu_table`, independent of the `asu_mappings` object; hence the naming. `pair_sym_table` is therefore suitable for communicating connectivity between algorithms that may require different `asu_mapping` objects due to shifts in coordinates or modified distance cutoffs. Here is how we can re-generate a new `pair_asu_table` from a `pair_sym_table`:

```
new_asu_mappings = quartz_structure.asu_mappings(buffer_thickness=5)
new_pair_asu_table = crystal.pair_asu_table(asu_mappings=new_asu_mappings)
new_pair_asu_table.add_pair_sym_table(sym_table=pair_sym_table)
new_pair_asu_table.show()
```

Output:

```
i_seq: 0
  j_seq: 1
    j_syms: [0, 3, 55, 68]
i_seq: 1
  j_seq: 0
    j_syms: [0, 8]
```

In this case the `j_syms` have changed compared to the output of `pair_asu_table.show()` because the buffer region of `new_pair_asu_table` is larger compared to that of the initial `pair_asu_table`.
The new iotbx.show_distances command provides an easy to use interface to the core functionality described in this section. This command reads files in the simple format introduced by the `kriber` program (http://www.crystal.mat.ethz.ch/Software/Kriber). For example:

```
*quartz

P 62 2 2
 5.01 5.47
Si   0.5000  0.5000  0.3333
O    0.1970 -0.1970  0.8333
---------------------------
```

The full command is:

```
iotbx.show_distances quartz_structure --distance_cutoff=1.7
```

Output:

```
strudat tag: quartz

Number of scatterers: 2
At special positions: 2
Unit cell: (5.01, 5.01, 5.47, 90, 90, 120)
Space group: P 62 2 2 (No. 180)
Label, Scattering, Multiplicity, Coordinates, Occupancy, Uiso
Si   Si      3 ( 0.5000  0.5000  0.3333) 1.00 0.0000
O    O       6 ( 0.1970 -0.1970  0.8333) 1.00 0.0000
```

```
Si(1):          pair count:   4  <<  0.5000,  0.5000,  0.3333>>
  O(2):        1.6160            (  0.1970,  0.3940,  0.5000)
  O(2):        1.6160 sym. equiv. (  0.3940,  0.1970,  0.1667)
  O(2):        1.6160 sym. equiv. (  0.8030,  0.6060,  0.5000)
  O(2):        1.6160 sym. equiv. (  0.6060,  0.8030,  0.1667)
O(2):           pair count:   2  <<  0.1970, -0.1970,  0.8333>>
  Si(1):       1.6160            (  0.0000, -0.5000,  0.6667)
  Si(1):       1.6160 sym. equiv. (  0.5000,  0.0000,  1.0000)

Pair counts: [4, 2]
```

The implementation of this command can be found in the file `iotbx/iotbx/command_line/show_distances.py`.

3.4: Nonbonded exclusions

In the refinement of macro-molecular structures it is common to use nonbonded pair interactions, e.g. Lennard-Jones potentials or empirical "repulsive force fields." For sites that are not bonded but are within a certain distance (typically around 7 Å) a corresponding nonbonded energy term is added to the total energy of the geometry. Experience shows that it is highly advantageous to exclude certain nonbonded interactions. Consider this simple molecular fragment:

```
A-B-C
    |
  E-D
```

The lines indicate bonded interactions. These are often referred to as "1-2" interactions. In our fragment we find 1-2 interactions between A-B, B-C, C-D, and D-E. The nonbonded interactions A-C, B-D, and C-E are commonly referred to as 1-3 interactions, and the nonbonded interaction A-D is called a 1-4 interaction. In general, 1-2 and 1-3 interactions are excluded from the nonbonded energy term, and 1-4 interactions are attenuated.

When setting up the nonbonded energy calculations we have to find the 1-3 and 1-4 interactions based on the pre-defined bonded (1-2) interactions. If space group symmetry is not involved this is very straightforward. However, if symmetry bonds are to be considered the situation becomes much more complex again. The algorithm required is known as "coordination sequence algorithm" and is commonly used in material science, in particular zeolite research (e.g. Brunner & Laves, 1971, Grosse-Kunstleve et al., 1996). See also the Atlas of Zeolite Framework Types available at http://www.iza-structure.org/).

3.5: cctbx.crystal.coordination_sequence

It is surprisingly easy to write a complete coordination sequence algorithm based on the pair_asu_table object discussed before. A `simple_and_slow` reference implementation can be found in the `cctbx.crystal.coordination_sequences` module. The complete function comprises just 36 lines of Python code. We can use this short function to easily compute the coordination sequences for the Si and O atoms in our `quartz_structure`:

```
import cctbx.crystal.coordination_sequences
term_table = crystal.coordination_sequences.simple_and_slow(
  pair_asu_table=pair_asu_table,
  max_shell=10)
for terms in term_table:
  print terms
```

Output:

```
[1, 4, 4, 12, 12, 36, 30, 84, 52, 124, 80]
[1, 2, 6, 6, 18, 18, 51, 42, 103, 62, 156]
```

The first list of terms is for the Si atom, the second for the O atom. The first term (in "shell" 0) is always 1. The 1 Si is bonded to 4 O atoms (shell 1), which are bonded to 4 new Si atoms (shell 2). Following all the bonds from these Si atoms to the next O atoms leads to 12 new O atoms (shell 3), from there to 12 new Si atoms (shell 4), etc.

In the mathematics of coordination sequences (e.g. Grosse-Kunstleve et al., 1996) it is most natural to index the coordination shells in the way shown above. Unfortunately this is not directly compatible with the nomenclature of 1-2, 1-3, and 1-4 interactions used in the macro-molecular field. The interactions accounted for in shell 1 are the 1-2 interactions, shell 2 accounts for the 1-3 interactions, and shell 3 for the 1-4 interactions.

To find the nonbonded exclusions we do have to do a little more work than just counting the number of interactions as is done by the `simple_and_slow` function. For each shell we have to keep a table of the interactions found. A much faster, optimized C++ implementation of the coordination sequence algorithm with interfaces for both simple counting and the generation of interaction tables is available in the file `cctbx/include/cctbx/crystal/coordination_sequences.h`. The Python interface to the simple and fast counting algorithm is very similar to that for the `simple_and_slow` interface:

```
term_table = crystal.coordination_sequences.simple(
  pair_asu_table=pair_asu_table,
  max_shell=10)
crystal.coordination_sequences.show_terms(
  structure=quartz_structure,
  term_table=term_table)
```

Output:

```
Si [1, 4, 4, 12, 12, 36, 30, 84, 52, 124, 80]
O [1, 2, 6, 6, 18, 18, 51, 42, 103, 62, 156]
TD10: 456.33
```

Here we make use of the `show_terms` function which shows the scatterer labels along with each list of terms and also the `TD10`, a measure of the "topological density" commonly used in the zeolite field (see http://www.iza-structure.org/).

The tabulation of the 1-3 and 1-4 interactions needed for the nonbonded exclusions is equally easy:

```
shell_asu_tables = crystal.coordination_sequences.shell_asu_tables(
  pair_asu_table=new_pair_asu_table,
  max_shell=3)
print shell_asu_tables
```

Output:

```
(<cctbx_crystal_ext.pair_asu_table object at 0x82a2bec>,
 <cctbx_crystal_ext.pair_asu_table object at 0x82a2c34>,
 <cctbx_crystal_ext.pair_asu_table object at 0x82a2c7c>)
```

The result is a Python tuple with three `pair_asu_table` objects for the 1-2, 1-3, and 1-4 interactions. The first `pair_asu_table` in the tuple is simply a reference to the original `pair_asu_table` defining the bonds. Keeping the original table together with the derived tables simplifies subsequent algorithms.

As an aside, the 36 lines of the `simple_and_slow` Python function have turned into 278 lines of C++ code in `coordination_sequences.h`. The size comparison is not quite fair because the C++ implementation works for both simple counting and tabulation of nonbonded exclusions, but a doubling or tripling of the lines of source code in the conversion from a Python reference implementation to the final C++ implementation is the norm. Unfortunately this is what we have to cope with until higher-level languages with smarter optimizers are a reality.

4: Putting everything together: cctbx.geometry_restraints.manager

The `shell_asu_tables` object of the previous section is the key data structure for the computation of the bond proxies and the nonbonded proxies. However, there is still much more to manage: we need to define the bond parameters (ideal distances and weights), nonbonded "energy types" and VdW (Van der Waals) distances, angle, dihedral, chirality and planarity restraints. Clearly we need a professional manager. It is implemented in the `cctbx.geometry_restraints.manager` module. The `manager` constructor acts as a tool for grouping all information required for the geometry restraints calculations:

```
class manager:

  def __init__(self,
        crystal_symmetry=None,
        site_symmetry_table=None,
        bond_params_table=None,
        shell_sym_tables=None,
        nonbonded_params=None,
        nonbonded_types=None,
        nonbonded_distance_cutoff=5,
        nonbonded_buffer=1,
        angle_proxies=None,
        dihedral_proxies=None,
        chirality_proxies=None,
        planarity_proxies=None):
```

A self-contained, reasonably simple example (266 lines of Python) for setting up all data structures for the bonded and nonbonded calculations can be found in the file `cctbx/cctbx/geometry_restraints/distance_least_squares.py`. This script performs a distance least squares minimization of zeolite geometries. It was developed primarily as a regression test, but covers almost all the functionality of the pioneering DLS-76 program (http://www.crystal.mat.ethz.ch/Software/DLS76). The only major DLS-76 feature missing is the refinement of unit cell parameters. The new `iotbx.distance_least_squares` command provides a simple interface to the script. In our internal test we use this command to minimize the geometries of the complete Atlas of Zeolite Framework Types (152 structures) in less than 40 seconds (Linux, Xeon 2.8GHz). This includes the automatic search for Si-Si bonds, the generation of oxygen atoms at the mid-points of the Si-Si bonds, the generation of angle restraints which are parameterized as pseudo O-O and Si-Si bonds, the generation of nonbonded interactions, and a two-stage minimization, first without a repulsive force field and in the second pass with the repulsions turned on. The successful completion of these minimizations gives us a high confidence that our system for the refinement of bonded and nonbonded pair interactions is complete and free of errors.

4.1: angle, dihedral, chirality, planarity restraints

The angle, dihedral, chirality, and planarity restraints are currently implemented in the "simple" version only, without treatment of symmetry. For our purposes this is fully sufficient and it may even be sufficient in general because angle restraints for small-molecule crystallography are often parameterized as pseudo bonds (e.g. DLS-76, see previous section). The three other restraint types are not very common in small-molecule crystallography. However, our framework is very open and symmetry-aware restraint types could probably be added without disturbing the overall organization of the cctbx.geometry_restraints module. To give an example we show how to work with angle restraints:

```
from cctbx import geometry_restraints
angle = geometry_restraints.angle(
  sites=[(1,2,3),(2,3,4),(5,4,3)],
  angle_ideal=120,
  weight=1)
print "angle_model:", angle.angle_model
print "delta:", angle.delta
print "residual:", angle.residual()
print "gradients:", angle.gradients()
```

Output:

```
angle_model: 121.482154105
delta: -1.48215410529
residual: 2.19678079184
gradients: ((-69.337848889979, 1.2765767806090013e-14, 69.337848889979028),
            (63.034408081799093, -25.213763232719657, -113.4619345472384),
            (6.3034408081799089, 25.213763232719643, 44.124085657259371))
```

Comparison with the first example for defining a bond restraint shows that the interfaces are very similar. Essentially we just need three `sites` instead of two, and we have to write `angle` everywhere instead of `bond` and `distance`. The higher level support for proxies, arrays of proxies and vector operations on these arrays is also very similar. The similarities extend to dihedral and chirality restraints where we need to specify four `sites` instead of two or three. Planarity restraints are slightly different because we have to deal with a variable number of sites and each site is associated with an individual weight:

```
from cctbx import geometry_restraints
from cctbx.array_family import flex
sites_cart = flex.vec3_double([
  (-6.9, 1.3, -1.4),
  (-4.9, -1.0, 0.1),
  (-6.9, -0.6, -1.7),
  (-4.8, 0.9, 0.5)])
weights = flex.double([1, 2, 3, 4])
planarity = geometry_restraints.planarity(
  sites=sites_cart,
  weights=weights)
print "deltas:", list(planarity.deltas())
print "residual:", planarity.residual()
print "gradients:", list(planarity.gradients())
```

We don't show the rather uninteresting output. The difference to the other restraint types is that we get an array of `deltas` instead of just one value. However the important `residual()` and `gradient()` functions fit into the common framework.

5: Setting up restraints using the CCP4 Monomer Library

The CCP4 (http://www.ccp4.ac.uk/) Monomer Library is a comprehensive database of protein, nuclear acid and hetero-compound geometries. We are grateful for CCP4 to give us permission to use this library. The new `mmtbx` top-level module of the `cctbx` project (see Newsletter No. 1 for information on the overall organization of the cctbx project) includes functions for reading the monomer library files as distributed by CCP4, and to generate the geometry proxies introduced above for a given PDB file (http://www.rcsb.org/). The end result is a `cctbx.geometry_restraints.manager.manager` instance that is completely independent of the Monomer Library, the PDB file, or any other file format. The `manager` object is then used in the same minimization procedure employed by the `distance_least_squares.py` script introduced before (the minimizer is implemented in `cctbx.geometry_restraints.lbfgs`). This complete separation of file formats and core computations makes it possible to support any other library defining geometry restraints. E.g. for the future we are planning to add support for CNS (http://cns.csb.yale.edu/) topology and parameter files.

Currently the code for working with the CCP4 Monomer Library resides in the `mmtbx/mmtbx/stereochemisty` directory. It is still being worked on in order to cleanly support PDB files with alternate conformations and it may be moved to a different place. We will describe the final result in the next newsletter.

6: Bulk solvent correction and scaling

It is well known that macromolecular crystals contain a large amount of disordered solvent reaching sometimes more than 70% of the unit cell volume. The scattering contribution of this solvent level becomes significant at low resolution starting from about 6.0 Å. There are several aspects where the appropriate modeling of low resolution data is of great importance: electron density map analysis (Urzhumtsev, 1991), crystallographic refinement (Kostrewa, 1997), precise calculation of electrostatic properties of molecules (Lecomte, 1999), and the translation search part of structure solution by the Molecular Replacement method (Fokine & Urzhumtsev, 2002a). Basically two bulk solvent models are currently in use by popular crystallographic packages: the exponential scaling model (Moews & Kretsinger, 1975; Tronrud, 1997) and the flat model (Phillips, 1980; Jiang & Brunger, 1994). The exponential scaling model is only justified for the very low-resolution data, lower than 15 Å (Podjarny & Urzhumtsev, 1997), and becomes incorrect at higher resolutions. The flat model is shown as physically more reasonable (Fokine & Urzhumtsev, 2002b) and being compared to all others models is demonstrated as more efficient in sense of both computations and quality of final result obtained (Jiang & Brunger, 1994).

Based on the arguments above, we implemented the flat bulk solvent model in the `mmtbx.bulk_solvent` module. The bulk solvent modeling and scaling procedure contains four main steps: molecule mask calculation, structure factors calculation from the mask, determination of solvent parameters ksol and Bsol, and determination of the overall anisotropic scale coefficient (Sheriff & Hendrickson, 1987).
The algorithm for the mask calculation is realized as described by (Jiang & Brunger, 1994). The corresponding Python code looks like this:

```
from mmtbx.bulk_solvent import bulk_solvent_models
from mmtbx.masks import masks
from iotbx import reflection_file_reader
from iotbx import pdb
pdb_file = "1F8T.pdb"
hkl_file = "1F8T.hkl"
xray_structure = pdb.as_xray_structure(pdb_file)
refl = reflection_file_reader.any_reflection_file(file_name=hkl_file)
refl_arrays = refl.as_miller_arrays(crystal_symmetry=xray_structure)
f_obs = refl_arrays[0].resolution_filter(d_min=2.5)
f_calc = f_obs.structure_factors_from_scatterers(
  xray_structure=xray_structure).f_calc()
mask_manager = masks.mask_utils(
  structure=xray_structure,
  mask_grid_step=f_obs.d_min()/4.,
  shell=5.0,
  shrink=1.0,
  rsolv=1.0)
f_mask = mask_manager.sf_from_mask(f=f_obs)
f_mask.set_info("Mask structure factors")
f_mask.show_summary()
print "Accessible surface fraction:", \
  mask_manager.accessible_surface_fraction()
print "Contact surface fraction:", \
  mask_manager.contact_surface_fraction()
```

Output:

```
Miller array info: Mask structure factors
Observation type: None
Type of data: complex_double, size=15897
Type of sigmas: None
Number of Miller indices: 15897
Anomalous flag: 0
Unit cell: (72.24, 72.01, 86.99, 90, 90, 90)
Space group: P 21 21 21 (No. 19)
Accessible surface fraction: 0.330885416667
Contact surface fraction: 0.45850308642
```

The bulk solvent structure factors and parameters ksol and Bsol can be calculated by adding the following lines to the previous code:

```
bulk_solvent_manager = bulk_solvent_models.bulk_solvent(
  verbose=-1,
  f_obs=f_obs,
  f_calc=f_calc,
  f_mask=f_mask,
  aniso_scale_flag=0001,
  bulk_solvent_correction_flag=0001)
print "Flat model bulk solvent parameters: ", \
  bulk_solvent_manager.ksol_bsol()
f_bulk = bulk_solvent_manager.f_bulk()
f_bulk.set_info("Bulk solvent structure factors")
f_bulk.show_summary()
```

Output:

```
Flat model bulk solvent parameters:  (0.31000000000000011, 38.0)
Miller array info: Bulk solvent structure factors
Observation type: None
Type of data: complex_double, size=15897
Type of sigmas: None
Number of Miller indices: 15897
Anomalous flag: 0
Unit cell: (72.24, 72.01, 86.99, 90, 90, 90)
Space group: P 21 21 21 (No. 19)
```

All major refinement programs use minimizers to determine the bulk solvent parameters and the anisotropic scaling matrix. However there are a number of difficulties to this approach:

1. The low-resolution diffraction data may not be of sufficient quality or completeness.
2. The starting values for ksol and Bsol may be far from the correct values.
3. The parameters ksol and Bsol are highly correlated. Therefore the minimizer may have difficulties finding a path to the global minimum.
4. Optimizing a function of two exponentials is generally a difficult problem.

These considerations have lead us to choose a more robust procedure. As was demonstrated by Fokine & Urzhumtsev (2002b), the values for ksol and Bsol are distributed around 0.35 eÅ^{-3} and 46 Å^{2}, respectively. Therefore we decided to implement a grid search procedure for the determination of ksol and Bsol. The search is conducted in a physically meaningful range of values, [0,1] for ksol and [0,100] for Bsol. For each trial pair (ksol, Bsol) in the specified ranges we calculate the anisotropic scaling

coefficients using a gradient-driven minimizer. Finally we select the values ksol and Bsol based on the best outcome of the minimization. It should be emphasized that we use the whole resolution range of data. In contrast, Jiang & Brunger (1994) suggest a partitioning into low-resolution and high-resolution pools. This is the approach used by the CNS program (Brunger et al., 1998) to make the minimization procedure more stable. Our grid-search procedure is sufficiently robust to work without partitioning the data.
Another new feature that distinguishes our implementation of the scaling procedure from previous implementations is the use of a maximum-likelihood function as the objective function in the minimization. Even though maximum-likelihood based refinement is now very common, all existing programs use a conventional least-squares target in the determination of the bulk solvent and scaling parameters, while maximum-likelihood functions are used to determine all other parameters. This inconsistency is eliminated in the `mmtbx.bulk_solvent` module.

7: Relative scaling of crystallographic functional and restraints

Crystallographic refinement usually considers the minimization of a sum of two functions. One function is responsible for fitting the model to the experimental data and the second function introduces restraints encoding a priori knowledge, for example the geometry restraints discussed before. The two functions are generally on a different scale and it is necessary to determine an appropriate relative scale factor in order to balance the contributions to the sum. For this purpose we have implemented the procedure proposed by (Adams et al., 1997) in the `mmtbx.refinement.weight_xray_term` module, which makes use of the new `mmtbx.dynamics` module.

8: Crystallographic target functions

The new `mmtbx.refinement` module implements two crystallographic target functions in addition to the conventional least-squares and correlation target functions provided by the `cctbx.xray` module. These are the full maximum-likelihood function of Lunin et al. (2002) and its quadratic approximation (Lunin & Urzhumtsev, 1999). The calculation of the distribution parameters for the target function, "alpha" and "beta", is implemented in two ways:

- estimation by maximization of a likelihood function given a current model and observed intensities (Lunin & Skovoroda, 1995),
- determination via an exact formula (see, for example, Afonine et al., 2003).

In the future we will also implement the sigma-a algorithm and likelihood functions including experimental phase information.

9: Integration of Clipper

Thanks to generous support by Kevin Cowtan, the Clipper library (http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper.html, see also Cowtan (2003) in Newsletter No. 2) is now integrated into the cctbx project and redistributed with the cctbx bundles posted at http://cci.lbl.gov/cctbx_build/ . The bundles with the build tag 2004_07_06_0816 are the first to include Clipper. Currently the Clipper libraries requiring fast Fourier transforms are not compiled in the cctbx build, but this is likely to change in the future. The supporting clipper_adaptbx adaptor toolbox in the cctbx tree provides a fully functional Python interface to the sigma-a calculations in Clipper. We will add other Python interfaces as the need arises.

10: Efficient sampling of search spaces

Indirectly Kevin Cowtan has left his mark in the cctbx project in another way. The newly added `cctbx.crystal.close_packing` module implements a hexagonal close packing sampling generator as suggested by Kevin for some time. Sampling space at the points of a hexagonal close packing instead of the points of a regular grid leads to significant speed increases in search procedures such as the molecular replacement translation search or the placement of molecular fragments in electron density maps. The `cctbx.crystal.close_packing.hexagonal_sampling` generator produces points to efficiently sample search spaces with various symmetries. Space group symmetry and Euclidean normalizer symmetry (also known as Cheshire symmetry) can be arbitrarily combined to define the symmetry of the search space. Depending on the settings, the resulting sampling points may cover three, two, one or zero dimensions. The symmetry is controlled at a high level via flags. The search-space symmetry operations including continuous allowed origin shifts are determined automatically. For example:

```
from cctbx import crystal
import cctbx.crystal.close_packing
from cctbx import sgtbx
crystal_symmetry = crystal.symmetry(
  unit_cell="255.260  265.250  184.400  90.00  90.00",
  space_group_symbol="P 21 21 2")
for use_space_group_symmetry in [True, False]:
  sampling_generator = crystal.close_packing.hexagonal_sampling(
    crystal_symmetry=crystal_symmetry,
    symmetry_flags=sgtbx.search_symmetry_flags(
      use_space_group_symmetry=use_space_group_symmetry,
      use_seminvariants=True,
      use_normalizer_k2l=False,
      use_normalizer_l2n=False),
    point_distance=2)
  print "number of sampling points:", sampling_generator.count_sites()
```

Output:

```
number of sampling points: 46332
number of sampling points: 162240
```

The whole procedure takes 0.14 seconds (Linux, Xeon 2.8GHz). See also the reference documentation for the classes `cctbx::crystal::close_packing::hexagonal_sampling_generator` and `cctbx::sgtbx::search_symmetry_flags` available at http://cctbx.sourceforge.net/ under "C++ interfaces."

11: Acknowledgments

We like thank the CCP4 project for giving us permission to use the Monomer Library, Kevin Cowtan for giving us permission to use Clipper, and the mmLib (http://pymmlib.sourceforge.net/) development team (E. Merritt and J. Painter) for giving us permission to use the `mmCIF.py` file. We gratefully acknowledge the financial support of NIH/NIGMS. Our work was supported in part by the US Department of Energy under Contract No. DE-AC03-76SF00098.

12: References

Adams, P. D., Pannu, N. S., Read, R. J. & Brunger, A. T. (1997). Proc. Natl. Acad. Sci. USA, 94, 5018-5023.
Afonine, P., Lunin, V. & Urzhumtsev, A. (2003). J. Appl.Cryst., 36,158-159.
Brunger, A. T., Adams, P. D., Clore, G. M., DeLano, W. L., Gros, P., Grosse-Kunstleve, R. W., Jiang, J.-S., Kuszewski, J., Nilges, M., Pannu, N. S., Read, R. J., Rice, L. M., Simonson, T. & Warren, G. L. (1998). Acta Cryst. D54, 905-921.
Brunner, G.O. & Laves, F. (1971). Wiss. Z. Techn. Univers. Dresden 20, 387-390.
Cowtwn, K. (2003). Newsletter of the IUCr Commission on Crystallographic Computing, 2, 4-9.
Fokine, A. & Urzhumtsev, A. (2002a). Acta Cryst., A58, 72-74.
Fokine, A. & Urzhumtsev, A. (2002b). Acta Cryst., D58, 1387-1392.
Grosse-Kunstleve, R.W., Brunner G.O., Sloane N.J.A. (1996). Acta Cryst. A52, 879-889.
Grosse-Kunstleve, R.W., Adams, P.D. (2003). Newsletter of the IUCr Commission on Crystallographic Computing, 1, 28-38.
Grosse-Kunstleve, R.W., Wong, B., Adams, P.D. (2003). Newsletter of the IUCr Commission on Crystallographic Computing, 2, 10-16.
Hendrickson, W.A. (1985). Meth. Enzym. 115, 252-270.
Jiang, J.-S. & Brunger, A.T. (1994). J. Mol. Biol. 243, 100-115.
Kostrewa, D. (1997). CCP4 Newsl. 34, 9-22.
Lecomte, C. (1999). Implications of Molecular and Materials Structure for New Technologies, NATO ASI and Euroconference, Serie E: Applied Science, pp. 23-44. Dordrecht: Kluwer.
Lunin, V.Y., Afonine, P.V. & Urzhumtsev, A. (2002). Acta Cryst. A58, 270-282.
Lunin, V.Y. & Skovoroda, T.P. (1995). Acta Cryst. A51, 880-887.
Lunin, V.Y., Urzhumtsev, A.G. (1999). CCP4 Newsletter on Protein Crystallography, 37, 14-28.
Moews, P.C. & Kretsinger, R.H. (1975). J. Mol. Biol. 91, 201-228.
Phillips, S.E.V. (1980). J. Mol. Biol. 142, 531-554.
Podjarny, A.D., Urzhumtsev, A.G. (1997). In Methods in Enzymology, Academic Press, San Diego., C.W.Carter, Jr., R.M.Sweet, eds. 276A, 641-658.
Sheriff, S. & Hendrickson, W.A. (1987). Acta Cryst. A43, 118-121.
Tronrud, D.E. (1997). Methods Enzymol. 277, 306-319.
Urzhumtsev, A. (1991). Acta Cryst. A47, 794-801.

Geometrically Restrained INorganic Structure Prediction : GRINSP

Armel Le Bail,
Université du Maine, Laboratoire des Fluorures, CNRS UMR 6010, Avenue O. Messiaen, 72085 Le Mans 9, France - Email : alb@cristal.org ; WWW: http://www.cristal.org/

Introduction

With the idea in mind that the structure of a compound like τ-AlF_3 [1], unique example of a new 3D 6-connected network (no other isostructural MX_3 known up to now), should have been predicted before the laborious structure determination succeeded (finalized in 1992, from powder diffraction data), one can decide to write the prediction software. Which language would be more appropriate ? This may depend on the algorithm retained, but generally it mainly depends on the scientist knowledge. Would it be reasonable to propose the subject for a thesis or a post-doc ? Subject : "You will have to demonstrate that MX_3 compounds, built up from [MX_6] regular octahedra, sharing exclusively corners, can be predicted just as zeolites are predictable". It would be a bit difficult to find the good candidate. Conclusion, you will have to do it by yourself. It is your idea after all... The programming language will be the language which you best know, appropriate or not : Fortran77 in that case. A programming language may be considered as not obsolete if a compiler still exists for building the executable on recent computers. The project, imagined in 1992, was frozen up to the end of 2003, till personal computers became fast enough (the frequency of microprocessors increased by a factor 100 in the 1992-2003 range, the number of transistors increased as well a lot) for tempting to solve the problem by a Monte Carlo approach.

When the realization of such a project starts, you are not even sure to succeed. Very recent publications show that, concerning 3D 4-connected nets, systematic enumeration is now based on advances in mathematical tiling theory [2-4]. Unless you are a brilliant mathematician, this may not be of a great help when trying to transpose to 3D 6-connected nets (so, it is verified again here that Monte Carlo is the solution retained by the illiterates...). Previous works on hypothetical zeolites were made by using classical physical model building [5] during the past 60 years, or simulated annealing [6]. Many recent works in inorganic structure prediction (as well as organic and organometallic) have produced huge quantities of hypothetical compounds (using commercial packages as CERIUS, etc), there is no room here for citing them all. But there was no systematic recent work on MX_3 compounds, apart from the famous book of Wells [7] about three-dimensional nets and polyhedra. Predictions of new MX_3 compounds are non-existent, if one excludes the obvious models built up by intergrowth of known structures (perovskite, Hexagonal Tungsten Bronze type - HTB, etc).

Algorithm

It was chosen to manage the Monte Carlo generation of 3D nets by using geometrical restraints established from the interatomic distances in known materials. So, this is absolutely not an *ab initio* approach of the structure prediction problem. Multiple difficulties were solved one after the other during the writing of the source code. A primarily version of the program named GRINSP (Geometrically Restrained INorganic Structure Prediction) [8] was limited to the building of tetrahedra linked by corners (3D N-connected nets with $N = 4$) and to the P1 space group, because this was the more easy for testing the feasibility, all being more simple to develop in P1 on the point of view of writing the code. Then, obtaining encouraging results (150 hypothetical zeolites built in P1), the project was generalized to various values of N (3, 4, 5, 6), and even to mixtures of M and M' cations with different coordinations, in any space group [8]. In a N-connected 3D net of M atoms, each M atom is connected to N other M atoms through X atoms, giving formulations MX_2 for $N = 4$ (tetrahedra connected by corners as in SiO_2 polymorphs and zeolites), MX_3 for $N = 6$ (octahedra connected by corners) or M_2X_3 for $N = 3$ (triangles sharing corners as in B_2O_3), etc. The X atoms have to lie at positions close to the mid point of two M atoms. Therefore, the key of the algorithm is to concentrate on the M atoms first. For such N-connected 3D nets, if a model shows all M atoms surrounded by the expected number of M atoms (3, 4, 5 or 6), then this model is a possible solution. This expected number of neighbours (`mvexp` below) is checked frequently in the GRINSP program code, each times a new set of M atoms is added either on a general or on a special position of a selected space group :

```
      C  We expect to have each M surrounded by 3, 4, 5 or 6 M at first
      C     M-M distances, then the total of expected neighbours is :
            mvexp=ncop(mcop(nl(1)))*itot
      C  See neighbouring -
      C  Are some M atoms neighbouring already completed ?
            call complet(itot,x,met,f,g,nv,mv,mv2tot,ncop,mcop,ibad)
      C  If mvexp-mv2tot = icon then store the result
            iresult=mvexp-mv2tot
            if(iresult.eq.icon)go to 1002
      C Place the next atom of type M at acceptable distances
      C       considering first and second M-M neighbours
```

The Fortran77 language is quite easy to understand for people speaking english. GRINSP contains a lot of comments (lines starting by C). Either calculations or text sequence manipulations and comparisons can be done (etc), you are limited only by your imagination. The subroutine `complet` in the lines of code above is too large for showing it all here. The full GRINSP code contains more than 3000 lines, only some parts will be selected and listed in this article.

In the purpose to obtain this adequation (`iresult=icon=0` above) between the espected number of neighbours and the calculated number, the model is built sequentially, adding one M atom after the other. GRINSP does not work by applying simulated annealing to a starting random configuration. Version 1.00 works schematically as follows, by using the Monte Carlo method :

- Manual selection of the restraints on cell parameters, of restrained interatomic distances, of the type(s) of coordinations, and of the space group. Then the Monte Carlo process starts.

- Random selection of the cell parameters inside of the predefined range. The random generator subroutine in GRINSP is `randi` (see below), returning a value between 0. and 1., called very often in the program; `nsym` = 1 is corresponding to the cubic case, `nsym` = 2 corresponds to tetragonal, `nsym` = 3 is hexagonal or trigonal, `nsym` = 4 is orthorhombic, etc, other variables (a, b, c, alp, bet, gam) are self-explicit for crystallographers :

```
C   Define the cell parameters
      a=(amax-amin)*randi(iseed)+amin
      if(nsym.eq.1)then
      b=a
      c=a
      go to 8000
      endif
      c=(cmax-cmin)*randi(iseed)+cmin
      if(nsym.eq.2.or.nsym.eq.3)then
      b=a
      go to 8000
      endif
      b=(bmax-bmin)*randi(iseed)+bmin
8000  continue
      if(nsym.le.5)then
      alp=90.
      bet=90.
      gam=90.
      if(nsym.eq.3)gam=120.
      if(nsym.eq.5)bet=betmin+betd*randi(iseed)
      go to 8500
      endif
      alp=alpmin+alpd*randi(iseed)
      bet=betmin+betd*randi(iseed)
      gam=gammin+gamd*randi(iseed)
8500  continue
```

- Random positioning of a first cation M (or M') of the future M_xX_y (or $M_xM'_yX_z$) compound on a general or special position, itself selected randomly.

```
C Place the first atom of first type (M1)
4502  itot=0
      nl(1)=1
      nl(2)=0
C Decide at random for the polyhedra type
      mcop(nl(1))=int(randi(iseed)*float(npol)+1.)
      if(mcop(nl(1)).gt.npol)mcop(nl(1))=npol
C Decide for the Wyckoff position selected between np1 and npos
      mwyc(nl(1))=int(randi(iseed)*float(npos-np1)+pp1)
C Decide for the atomic coordinates
      x0=randi(iseed)
      y0=randi(iseed)
      z0=randi(iseed)
      gen=gen+1.
C Extend to all positions corresponding to mwyc(nl(1))
      is=mwyc(nl(1))
      it0=itot+1
      it01=it0-1+nas(is)
      if(it01.gt.65)go to 4502
      DO 299 k=1,nas(is)
      itot=itot+1
      it1(itot)=it0
      it2(itot)=it01
      ist(itot)=is
      mcop(itot)=mcop(nl(1))
      x(itot,1,1)=x0*smt(is,k,1,1)+y0*smt(is,k,1,2)+z0*smt(is,k,1,3)
     1+tt(is,k,1)
      x(itot,1,2)=x0*smt(is,k,2,1)+y0*smt(is,k,2,2)+z0*smt(is,k,2,3)
     1+tt(is,k,2)
      x(itot,1,3)=x0*smt(is,k,3,1)+y0*smt(is,k,3,2)+z0*smt(is,k,3,3)
     1+tt(is,k,3)
299   continue
C  Now we have itot atoms already, but...
C  Avoid short distances
      if(itot.eq.1)go to 302
      do 300 mm1=1,itot-1
      do 300 mm2=it1(itot),itot
      if(mm1.eq.mm2)go to 300
        p1=abs(x(mm1,1,1)-x(mm2,1,1))
        q1=abs(x(mm1,1,2)-x(mm2,1,2))
        r1=abs(x(mm1,1,3)-x(mm2,1,3))
        if(p1.gt.0.5)p1=p1-1.
        if(q1.gt.0.5)q1=q1-1.
        if(r1.gt.0.5)r1=r1-1.
      rr=met(1,1)*p1*p1+met(2,2)*q1*q1+met(3,3)*r1*r1
     1+met(1,2)*p1*q1+met(1,3)*p1*r1+met(2,3)*q1*r1
        if(rr.lt.f(1,mcop(1))) go to 4502
300   continue
302   continue
```

- Random positioning of the next cations (random choice of M or M') in respect of the distance restraints with the previous ones, on a general or special position, itself selected randomly.

```
C  Select randomly a M atom for adding its next neighbour
3500  m=int(randi(iseed)*float(itot)+1.)
      memo=m
C  Decide first which positions would generate too much M atoms
C             and eliminate them
      call toomuch(itot,npos,nas,v,fdmax3,np2,np1,pp1)
      if(np2.eq.0)go to 5002
C Decide for the polyhedra type
      mcop(itot+1)=int(randi(iseed)*float(npol)+1.)
      if(mcop(itot+1).gt.npol)mcop(itot+1)=npol
C  Determine how many neighbours ? And according to that,
C  select the appropriate treatment :
C  1- if ncop(1)=4 then
C   if mv(m,1)=4  : atom already completed
C   if mv(m,1)=3, 2, or 1  : one atom to add in correct position
```

```
C  2- if ncop(1)=6 then
C   if mv(m,1)=6  : atom already completed
C   if mv(m,1)=5, 4, 3, 2, or 1 : one atom to add in correct position
      if(ncop(mcop(m)).eq.3)go to (3001,3002,3500)mv(m,1)
      if(ncop(mcop(m)).eq.4)go to (3001,3002,3003,3500)mv(m,1)
      if(ncop(mcop(m)).eq.5)go to (3001,3002,3003,3004,3500)mv(m,1)
      if(ncop(mcop(m)).eq.6)go to (3001,3002,3003,3004,3005,3500)mv(m,1)
C  Case with only one previous neighbour
3001  continue
     gen22=0.
C  Decide for the Wyckoff position selected between np1 and npos
600   mwyc(itot+1)=int(randi(iseed)*float(npos-np1)+pp1)
      p=(x(m,1,1)-xa)+xa2*randi(iseed)
      q=(x(m,1,2)-xb)+xb2*randi(iseed)
      r=(x(m,1,3)-xc)+xc2*randi(iseed)
      gen=gen+1.
      gen22=gen22+1.
      if(gen.gt.genmax)go to 5002
C Extend to all positions corresponding to mwyc(nl(1))
........ Etc.
C  Case with already 2 previous neighbours
3002  continue
........ Etc
C  Case with already 3 neighbours
3003  continue
........ Etc
```

- If a model fulfills all distance restraints, place the X atoms at M-M midpoints, refine the atomic positions and cell parameters so as to improve an R factor (called `Rdt` below).

```
C     Place the X atoms and then refine by Monte Carlo
      imemnl=nl(1)
      nl(1)=itot
      call midpt(x,nv,mv,nl,ncop,mcop)
............Etc.
C  Monte Carlo distance improvement
C    Loop of idls moves per atom
C            but move also the cell parameters
      improve=0
      imove=0
      nltot=nl(1)+nl(2)
      mc=idls*nltot
      do 7500 imc=1,mc
C  Select an atom
C          i for type 1 or 2
C          m for atom order in the list of either nl(1) or nl(2)
C  or select one cell parameter (icel=1)
      icel=int(randi(iseed)*2.)
C  Do not refine the cell if iref = 0
      if(iref.eq.0)icel=0
C     change a cell parameter a or b or c by (+ or -) 0.01 A max
C     change an angle cell parameter alp, bet or gam
C                             by (+ or -) 0.01 ° max
      if(icel.eq.1)then
      if(nsym.le.4)mc=int(randi(iseed)*3.+1.)
      if(nsym.eq.5)mc=int(randi(iseed)*4.+1.)
      if(nsym.eq.6)mc=int(randi(iseed)*6.+1.)
C  Redefine the cell parameters
      if(mc.eq.1)anew=a+(randi(iseed)-0.5)*0.02
      if(mc.eq.2)bnew=b+(randi(iseed)-0.5)*0.02
      if(mc.eq.3)cnew=c+(randi(iseed)-0.5)*0.02
      if(mc.eq.4)betnew=bet+(randi(iseed)-0.5)*0.02
      if(mc.eq.5)alpnew=alp+(randi(iseed)-0.5)*0.02
      if(mc.eq.6)gamnew=gam+(randi(iseed)-0.5)*0.02
      if(nsym.eq.1)then
      if(mc.eq.1)then
      bnew=anew
      cnew=anew
      endif
      if(mc.eq.2)then
      anew=bnew
      cnew=bnew
      endif
      if(mc.eq.3)then
      anew=cnew
      bnew=cnew
      endif
      alpnew=alp
      betnew=bet
      gamnew=gam
      go to 8001
      endif
      if(nsym.eq.2.or.nsym.eq.3)then
.......Etc
C Orthorhombic or more
      if(nsym.ge.4)then
C  If RdT improved, keep the move...
      rdtnew=sqrt((rd1new+rd2new+rd3new)/
     1(rd1dnew+rd2dnew+rd3dnew))
      if(rdtnew.ge.rdt)go to 7500
      improve=improve+1
C   Move kept, then make all changes...
C  changes accepted either on cell or atom moves
      if(icel.eq.1)then
C  Here modif on cell if accepted
      a=anew
      b=bnew
      c=cnew
      alp=alpnew
      bet=betnew
      gam=gamnew
      else
C   changes on coordinates x and Rd1,Rd2 and Rdt
      x(m,i,1)=pmc
      x(m,i,2)=qmc
      x(m,i,3)=rmc
C   changes on y
      if(i.eq.1)then
      do ki=1,27*nl(1)
      if(ny(ki).eq.m)then
      y(ki,1)=y(ki,1)+dp
      y(ki,2)=y(ki,2)+dq
      y(ki,3)=y(ki,3)+dr
      endif
       enddo
       endif
C   Also changes on xy if i=2...
      if(i.eq.2)then
      do ki=1,nl(1)
      do kj=1,ncop(mcop(ki))
      if(nxy(ki,kj).eq.m)then
      xy(ki,1,kj)=xy(ki,1,kj)+dp
      xy(ki,2,kj)=xy(ki,2,kj)+dq
      xy(ki,3,kj)=xy(ki,3,kj)+dr
      endif
      enddo
       enddo
       endif
C   changes on coordinates x and Rd1,Rd2,Rd3 and Rdt
      rd1=rd1new
      rd2=rd2new
      rd3=rd3new
      rdt=rdtnew
      endif
C      write(10,*)'i,m ',i,m
7500  continue
C  End of Monte Carlo distance improvement
C  If Rdt > Rdtmax, reject the cell
      if(rdt.gt.rdtm)then
      igood=igood-1
      go to 5001
      endif
C   Do not save if the framework density is outside
C        of the expected range
      if(v.lt.1.)then
      igood=igood-1
      go to 5001
      endif
      tn=nl(1)
      rho(igood)=tn/v*1000.
      if(rho(igood).lt.fdmin)then
      igood=igood-1
      go to 5001
      endif
      if(rho(igood).gt.fdmax)then
      igood=igood-1
      go to 5001
      endif
```

- Continue to try to predict structures in that way till a certain number of independent runs are made. Verify if the predicted structures are new or were already described (using CS - Coordination Sequence, a fingerprint of the structure).

```
C  Calculate coordination sequence
      call coorseq(nl,x,met,g,ntype,nr,ns,jmax,ncop,mcop,npol)
.......Etc
C  Try to identify if this is already known or already predicted
C     Now compare with data in connectivity.txt ...
C        prepare the current data
      iprint=1
      do i=1,ntype
      do j=1,10
      write(t(j),'(i4)')nr(i,j)
      enddo
      newcos(i,igood)=t(1)//t(2)//t(3)//t(4)//t(5)//t(6)//t(7)//
     1t(8)//t(9)//t(10)
      enddo
C     compare only on the real length of the predicted sequence
      l=4*jmax
      do 6800 i=1,ndat
C        skip if not same ntype
      if(nzeot(i).ne.ntype)go to 6800
      isum=0
      do 6700 j=1,ntype
      do 6699 k=1,nzeot(i)
      if(newcos(j,igood)(1:l).eq.coseq(k,i)(1:l))then
      isum=isum+1
      if(isum.eq.ntype)go to 6801
      endif
6699  continue
6700  continue
6800  continue
C    Nothing found
      go to 6810
C    Something found
6801  continue
      ident(igood)=zeot(i)
      write(10,*)
      write(10,*)'This is probably ',zeot(i)
      iprint=0
C  but make the output files .cif, .dat and .xtl anyway
C              if this is asked for (isave=1)
      if(isave.eq.1)iprint=1
      go to 6820
6810  continue
C  Compare also with current list
      do 6850 i=1,igood-1
C        skip if not same ntype
      if(ntype.ne.mntype(i))go to 6850
      isum=0
      jmaxmin=jmax
      if(mjmax(i).lt.jmaxmin)jmaxmin=mjmax(i)
      l=4*jmaxmin
      do 6750 j=1,ntype
      do 6749 k=1,mntype(i)
      if(newcos(j,igood)(1:l).eq.newcos(k,i)(1:l))then
      isum=isum+1
```

```
            if(isum.eq.ntype)then
            if(rd(igood).gt.rdp(i))iprint=0
            if(rd(igood).lt.rdp(i))rdp(i)=rd(igood)
            go to 6851
            endif
         go to 6750
         endif
   6749  continue
   6750  continue
   6850  continue
   C    Nothing found
         go to 6860
   C    Something found
   6851  continue
         ident(igood)=ident(i)
         write(10,*)
         write(10,*)'This was found already ',ident(i)
         go to 6820
   6860  continue
   C   If nothing found, this is a new one...
         inew=inew+1
         write(ident(igood),'(a4,i7)')'PCOD',inew
         write(10,*)
         write(10,*)'Found for the first time ',ident(igood)
   6820  continue
   C  End of checking
```

In the GRINSP algorithm, the number of M or M' atoms in a randomly selected cell is not predetermined, it is predicted as well. Only restraints on distances are considered (not angles - though considering a range for the second M-M distances is like restraining angles).

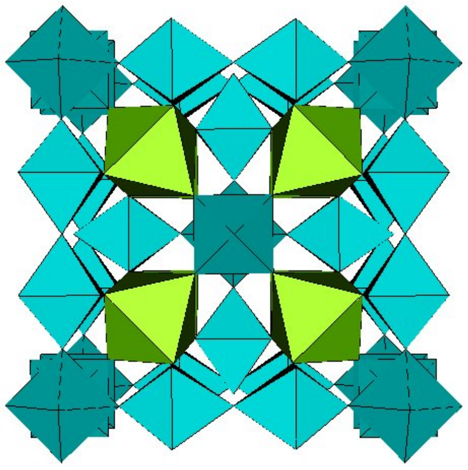
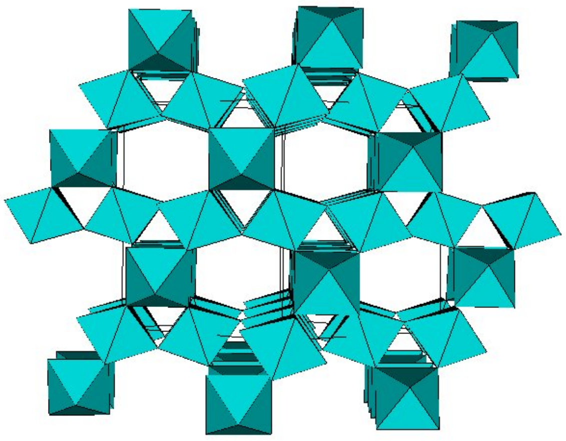
Currently, there are some limitations in that version. GRINSP 1.00 proved to be relatively efficient for a maximum number of 64 M/M' atoms on up to 1-10 different general or special positions. It was possible to retrieve many known zeolites (ABW, ACO, AFI, ANA, AST, BIK, CAN, EAB, EDI, GIS, GME, LOS, LTA, MEP, SOD...) and the compact SiO_2 phases (quartz, cristobalite, tridymite, etc), polymorphs for B_2O_3, AlF_3, hypothetical phases in binary systems B_2O_3/SiO_2, B_2O_3/ReO_3, SiO_2/ReO_3 (see the PCOD [9], a database accumulating these predicted structures). It is up to you to try GRINSP with other systems, and even the above ones have not been completely explored (in part due to that limitation to 64 M/M' atoms and because the maximum cell parameters were generally set to 16 Å). One life would not be sufficient if one formulation explored for one space group needs one or several days of calculations on a standard PC.

Further work is needed for improving the GRINSP efficiency : introduction of different linkage modes than by corners (edges, faces...) but this would mean that all X atoms could not be placed at the M-M mid points; adding the possibility for insertion of big cations K/Sr/Ba/Cs/etc as spheres in the holes/tunnels; considering bond valence as an alternative to pure geometrical restraints for the model final refinements; increase the speed by not recalculating always everything (distances); increase the box size for the CS (coordination sequence) calculations (the 729 cells used are not always enough); increase that 64 M/M' atoms limit; allow to select the space group randomly as well; optimize the code; etc !

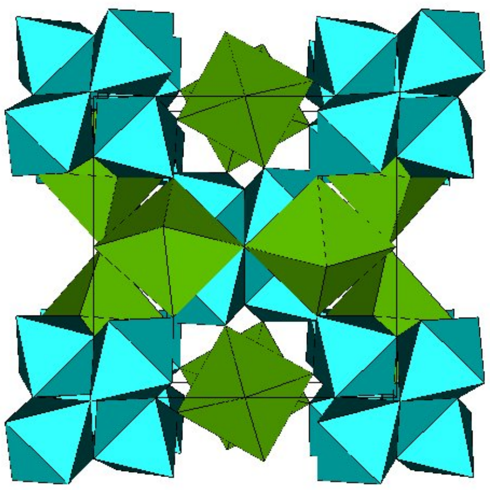
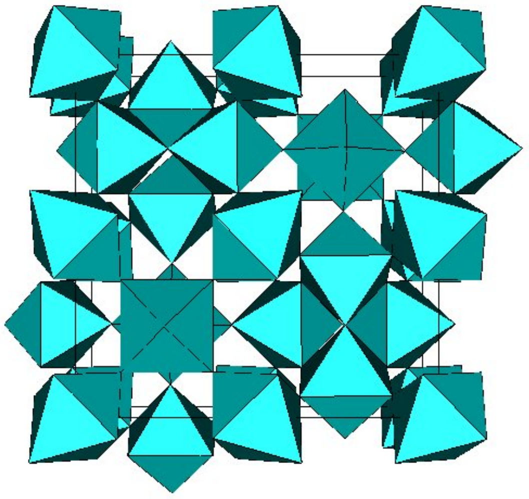
These improvements would need faster microprocessors or using a grid of computers on the internet. Anyway, the main problem is that once a model is built, it has to be checked visually. The process is not yet completely automatized (my confidence in it is not absolute, some two-dimensional models have to be discarded, etc).

A few results with 3D 6-connected frameworks

All the known varieties of AlF_3 (pyrochlore, perovskite, HTB...) are predicted by GRINSP, including this strange τ-AlF_3. New varieties are to be expected, if the GRINSP predictions are confirmed. Mixed compounds with two octahedra sizes were also modelled. Some are knowns, other are not. In some cases, the chemical composition is enough precise for suggesting the synthesis (contrarily to a simple MX_3 or MX_2 formula), see the figures below. Work is in progress for trying to confirm some of these predictions.

Figures 1 and 2: ***PCOD1000015*** *$[Ca_4Al_7F_{33}]^{4-}$, cubic, a = 10.860 Å. Known with Na atoms in the holes, as $Na_4Ca_4Al_7F_{33}$* ; ***PCOD1000017*** *AlF_3, cubic, a = 9.668 Å. Known with some OH^- and water in the holes : pyrochlore*

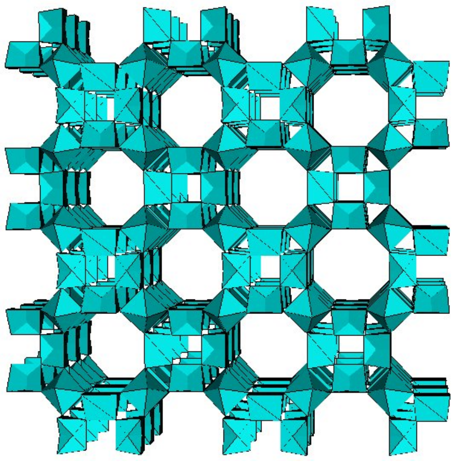



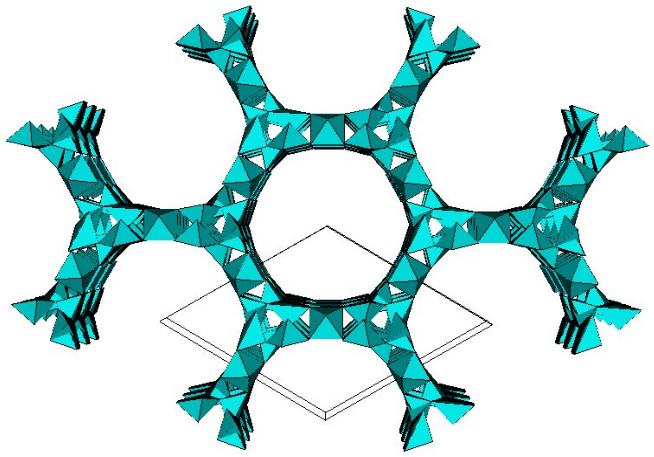
Figures 3 and 4: ***PCOD1010005*** *$[Ca_3Al_4F_{21}]^{3-}$, cubic, a = 9.009 Å.* ***UNKNOWN*** *Could be stuffed by Na atoms and give the hypothetical $Na_3Ca_3Al_4F_{21}$ - or stuffed by Li atoms ;* ***PCOD1000014*** *AlF_3, tetragonal, a = 10.216 Å, c = 7.241 Å. Known as t-AlF_3*

The above pictures are screen copies from drawings made by using a VRML visualizer (CosmoPlayer) reading the .wrl files stored in the PCOD. These .wrl files were made by the STRUVIR program from the .dat files also available in PCOD (as well as CIF files). GRINSP itself has no graphical user interface (GUI), which may not be absolutely necessary if one considers the simplicity of the data necessary for a prediction :

```
Test P-62c - 190            !  Title
P -6 2 C                    !  Space group
3 1 0 2                     !  Codes for cell symm., 1 or 2 M atom types, icon, min nber of M
4                           !  Coordination of the M atom(s)
Si  O                       !  Definition of the MX couple(s)
6. 16. 6. 16. 6. 16.        !  Min and max a, b, c
90. 90. 90. 90. 120. 120. !  Min and max angles
5. 30.                      !  Min and max framework density
200000 300000 0.02          !  Monte Carlo runs, Monte Carlo events per run, Rdtmax
10000 1                     !  Monte Carlo events for x,y,z refinement, cell refined or not
1900000                     !  First filename
```

Concerning the sixfold coordination, GRINSP can produce it randomly as octahedra, trigonal prisms or pyramids with a pentagonal base. Moreover, if the tolerance factor R (`Rdt` in the code) is above 1% , then these polyhedra can be more or less distorted. Some predictions are showing octahedra/trigonal prisms or octahedra/pyramids mixtures. Fancy predictions with large tunnels or huge cavities are sometimes proposed, such as these two examples (on the left, octahedra and trigonal prisms, on the right, octahedra and pyramids mixtures, the pentagonal base of the pyramids covering the tunnel walls) :





Conclusion

Structure prediction is certainly a promising approach, and an unavoidable part of our future in crystallography. It would have to be combined with an efficient prediction of the physical properties and, more difficult, with the prediction of a synthesis procedure... Storing and managing the huge quantity of hypothetical phases will be a problem, and the one structure/one publication scheme applied for the real compounds will not be relevant to predicted compounds. With 540 predicted structures from GRINSP, PCOD is a dwarf compared to a brand new database of hypothetical zeolites [10] containing in 3 parts (at the time of writing this paper) : 114010 structures in the Bronze database (raw predictions), 33652 refined structures in the Silver database, but nothing yet in the Gold database which will contain unique models. PCOD contains already unique models (almost), and is not restricted to the SiO_2 formulation. The number of SiO_2 predicted polymorphs in the PCOD continues to increase by the slow exploration of all space groups with GRINSP (only the triclinic and cubic space groups were examined systematically yet). The final number of structures depends on the limit fixed on R for retaining or not a model. The R value was arbitrarily chosen to be smaller than 1% in the SiO_2 case. This allows to produce the observed zeolites and the known dense SiO_2 phases, and will finally add a few thousands of hypothetical ones when the exploration will be completed. The MX_3 3D 6-connected hypothetical frameworks will certainly be much less numerous. The SiO_2/B_2O_3 system was found even richer than for the simple SiO_2 formulation, even limiting R below 0.6%, though there is not any $B_xSi_yO_z$ in the ICSD database which would include BO_3 triangles and SiO_4 tetrahedra interconnected by corners ! So, what to do with all these predictions now ?

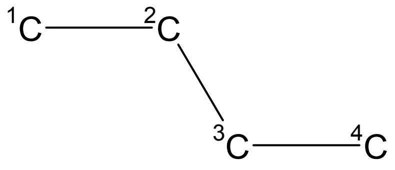
References

[1] A. Le Bail, J.L. Fourquet and U. Bentrup, *J. Solid State Chem*. **100** (1992) 151-159.
[2] O. Delgado Friedrichs, A.W.M. Dress, D.H. Huson,, J. Klinowski, A.L. Mackay, Nature 400 (1999) 644.
[3] M.D. Foster, O. Delgado Friedrichs, R.G. Bell, F.A. Almeida Paz, J. Klinowski, Angew Chem. Int. Ed. 42 (2003) 3896-3899.
[4] Systematic enumeration of crystalline networks : http://www-klinowski.ch.cam.ac.uk/SistEnumHome.htm
[5] J.V. Smith, Chem. Rev. 88 (1988) 149-182.
[6] M.W. Deem and J.M. Newsam, J. Am. Chem. Soc. 114 (1992) 7189-7198.
[7] A.F. Wells (1977). Three-dimensional Nets and Polyhedra (Wiley-Interscience, New York).
[8] GRINSP : http://sdpd.univ-lemans.fr/grinsp/
[9] PCOD - Predicted Crystallography Open Database: http://www.crystallography.net/pcod/
[10] Database of Hypothetical Zeolites : http://www.hypotheticalzeolites.net/

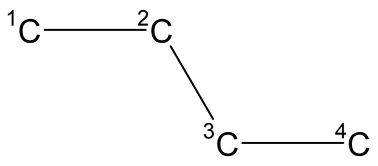
Whole molecule constraints - the Z-matrix unravelled

Kenneth Shankland,
ISIS Facility, Rutherford Appleton Lab., Oxon OX11 0QX, U.K. - Email : K.Shankland@rl.ac.uk ; WWW: http://www.isis.rl.ac.uk/dataanalysis/people/ken/KEN.HTM

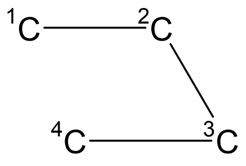
The success of global optimisation methods in the area of structure determination from powder diffraction data (SDPD) for organic molecules depends critically on the incorporation of prior chemical information. This information takes the form of the connectivity of the molecule under investigation; molecules are typically parameterised as a series of known bond lengths, known bond angles, and a mixture of known and unknown torsion angles. There are numerous ways of describing a molecule, but one of the most useful is the internal coordinate description [1] that is widely used in molecular modelling. As a picture of the molecule is built up atom-by-atom using a series of distance, angle and torsion specifications, it maps nicely to the requirements of the global optimisation problem. This is most easily seen with some examples. Consider a hypothetical molecule that consists solely of four sp^3 hybridised carbon atoms.



This molecule can be described by (*a*) an atom at an origin (^{1}C) with (*b*) a second atom (^{2}C) lying 1.54Å away from it, (*b*) an atom (^{3}C) at a distance of 1.54Å away from the ^{2}C, making an angle of 109.5º with the ^{2}C and ^{1}C and (d) a fourth atom (^{4}C) at a distance of 1.54Å away from the ^{3}C, making an angle of 109.5º with the ^{3}C and ^{2}C, and making a torsion (twist) angle with the ^{3}C, ^{2}C and ^{1}C. The torsion angle can take any value from 0 to 360°. Let us for a moment assume that the fourth carbon needs to lie in the same plane as the other three atoms. This limits the possible values to 0 and 180°, corresponding to the following configurations:



180°



0°

A convenient format for writing this description is a Z-matrix; here is a Z-matrix representation of the 180° configuration shown above:

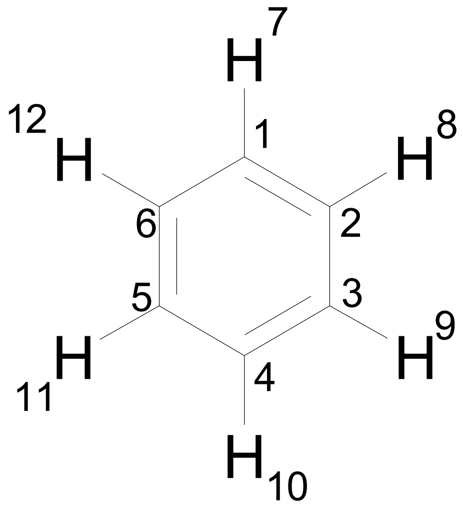
```
Atom      Distance Ref         Angle Ref      Torsion Ref    D     A     T
  C1     0.0000000  0     0.0000000  0     0.0000000  0     0     0     0
  C2     1.5400000  0     0.0000000  0     0.0000000  0     1     0     0
  C3     1.5400000  0   109.5000000  0     0.0000000  0     2     1     0
  C4     1.5400000  0   109.5000000  0   180.0000000  0     3     2     1
```

D = Atom with which the current atom makes a distance
A = Atom with which the current atom makes an angle, via the ‘D’ atom
T = Atom with which the current atom makes a torsion, via the ‘D’ and ‘A’ atoms
Ref = Refinement flag (0=norefine,1=refine, *see later*)

The Z-matrix can be read line-by-line in English, as follows:

1. Place atom #1, ‘C’, at 0,0,0
2. Place atom #2, ‘C’, at a distance of 1.54Å away from atom 1
3. Place atom #3, ‘C’, at a distance of 1.54Å away from atom 2, making an angle of 109.5° with atoms 2 and 1
4. Place atom #4, ‘C’, at a distance of 1.54 Å away from atom 3, making an angle of 109.5° with atoms 3 and 2, and making a torsion of 180° with atoms 3,2 and 1.

Any additional atoms are added in exactly the same way. It should now be obvious that the internal coordinate description allows one to easily describe an isolated molecule. However, the conformation of the molecule will be arbitrary, as one does not know *a-priori* what the values of certain torsion angles will be. Of course, there will be times when the molecule is completely defined by the Z-matrix e.g. benzene:



```
C1    0.0000000  0    0.0000000  0    0.0000000  0    0    0    0
C2    1.4000000  0    0.0000000  0    0.0000000  0    1    0    0
C3    1.4000000  0  120.0000000  0    0.0000000  0    2    1    0
C4    1.4000000  0  120.0000000  0    0.0000000  0    3    2    1
C5    1.4000000  0  120.0000000  0    0.0000000  0    4    3    2
C6    1.4000000  0  120.0000000  0    0.0000000  0    5    4    3
H7    1.0000000  0  120.0000000  0  180.0000000  0    1    2    3
H8    1.0000000  0  120.0000000  0  180.0000000  0    2    3    4
H9    1.0000000  0  120.0000000  0  180.0000000  0    3    4    5
H10   1.0000000  0  120.0000000  0  180.0000000  0    4    5    6
H11   1.0000000  0  120.0000000  0  180.0000000  0    5    6    1
H12   1.0000000  0  120.0000000  0  180.0000000  0    6    1    2
```

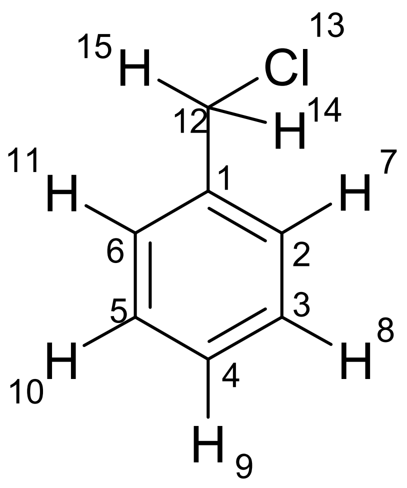
It should also be apparent from the above description that there are many different ways of defining a Z-matrix. One could start with a different atom at the origin, or, for example, H8 could have been described by:

```
H8    1.0000000  0  120.0000000  0  180.0000000  0    2    1    6
```

or

```
H8    1.0000000  0  120.0000000  0    0.0000000  0    2    1    7
```

Whilst not important for this particular example, it is very important when describing systems where torsional flexibility exists. For example, consider the following:



The CH_2Cl group can rotate around bond 1-12. If we assume for a moment that the chlorine atom lies in the plane of the ring, then one Z-matrix that would describe this molecule is;

```
 C1    0.0000000  0    0.0000000  0    0.0000000  0    0    0    0
 C2    1.4000000  0    0.0000000  0    0.0000000  0    1    0    0
 C3    1.4000000  0  120.0000000  0    0.0000000  0    2    1    0
 C4    1.4000000  0  120.0000000  0    0.0000000  0    3    2    1
 C5    1.4000000  0  120.0000000  0    0.0000000  0    4    3    2
 C6    1.4000000  0  120.0000000  0    0.0000000  0    5    4    3
 H7    1.0000000  0  120.0000000  0  180.0000000  0    2    3    4
 H8    1.0000000  0  120.0000000  0  180.0000000  0    3    4    5
 H9    1.0000000  0  120.0000000  0  180.0000000  0    4    5    6
 H10   1.0000000  0  120.0000000  0  180.0000000  0    5    6    1
 H11   1.0000000  0  120.0000000  0  180.0000000  0    6    1    2
 C12   1.4000000  0  120.0000000  0  180.0000000  0    1    2    3
 Cl13  1.7000000  0  109.5000000  0    0.0000000  0   12    1    2
 H14   1.0000000  0  109.5000000  0  120.0000000  0   12    1    2
 H15   1.0000000  0  109.5000000  0  240.0000000  0   12    1    2
```

Whilst this is satisfactory for this single conformation of the molecule, it is clear that if we want the ability to generate any permissible conformation about bond 1-12, we need to add an appropriate rotation angle to each of the last three torsions e.g. a 15 degree rotation about 12-1 would result in the last three lines of the Z-matrix being:

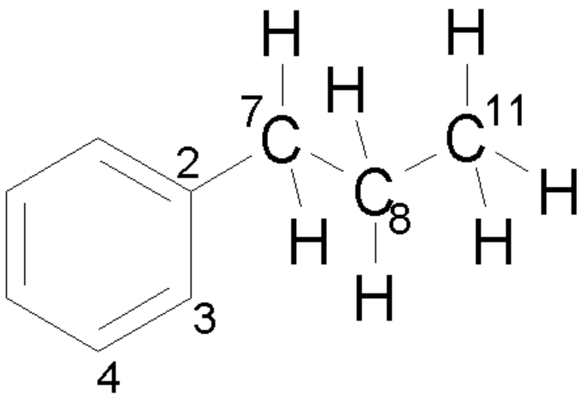
```
 Cl13  1.7000000  0  109.5000000  0   15.0000000  0   12    1    2
 H14   1.0000000  0  109.5000000  0  135.0000000  0   12    1    2
 H15   1.0000000  0  109.5000000  0  255.0000000  0   12    1    2
```

However, simply by changing the last three lines to:

```
 Cl13  1.7000000  0  109.5000000  0    0.0000000  1   12    1    2
 H14   1.0000000  0  109.5000000  0  120.0000000  0   12    1   13
 H15   1.0000000  0  109.5000000  0  120.0000000  0   12    1   14
```

we create a Z-matrix where any allowable value can be entered for torsion 13-12-1-2 and the attached hydrogen atoms will automatically rotate, as they are now defined *relative* to the position of the chlorine atom. Thus the chlorine atom makes a *proper* torsion with atoms 12,1,2, whilst the next hydrogen atom makes an *improper* torsion with atoms 12,1,13, and the last hydrogen makes an *improper* torsion with atoms 12,1,14. The 120° angles come from the fact that when we look along the 12-1 bond, we see three substituents coming off the sp^3 hybridised carbon atom, so the angles between these substituents must be 120°. Note also the presence of the '1' in the torsion refinement column, indicating that it is this torsion angle that will be varied in order to generate different conformations.

So in creating a Z-matrix that allows torsional rotations, we must take care to ensure that there is *only one proper torsion* in the Z-matrix for each of the torsion angles in the molecule that we want to vary. Here is another example, with the hydrogen atoms on the benzene molecule missed out for clarity.



```
C1    0.0000000  0    0.0000000  0    0.0000000  0    0    0    0
C2    1.4000000  0    0.0000000  0    0.0000000  0    1    0    0
C3    1.4000000  0  120.0000000  0    0.0000000  0    2    1    0
C4    1.4000000  0  120.0000000  0    0.0000000  0    3    2    1
C5    1.4000000  0  120.0000000  0    0.0000000  0    4    3    2
C6    1.4000000  0  120.0000000  0    0.0000000  0    5    4    3
C7    1.4000000  0  120.0000000  0  180.0000000  0    2    3    4
C8    1.5400000  0  109.5000000  0    0.0000000  1    7    2    3
H9    1.0000000  0  109.5000000  0  120.0000000  0    7    2    8
H10   1.0000000  0  109.5000000  0  240.0000000  0    7    2    8
C11   1.5400000  0  109.5000000  0  180.0000000  1    8    7    2
H12   1.0000000  0  109.5000000  0  120.0000000  0    8    7   11
H13   1.0000000  0  109.5000000  0  240.0000000  0    8    7   11
H14   1.0000000  0  109.5000000  0  180.0000000  0   11    8    7
H15   1.0000000  0  109.5000000  0  120.0000000  0   11    8   14
H16   1.0000000  0  109.5000000  0  240.0000000  0   11    8   14
```

Note again the use of a ‘1’ in the torsion refinement column, to indicate the relevant torsions we wish to vary. Hence in this molecule, we want rotations to occur around C7-C2 and C8-C7. Rotation around C11-C8 is ignored, because C11 has three equivalent ‘H’ atom substituents, and therefore we are unlikely to see any impact upon the calculated diffraction pattern as a result of a rotation around this bond. This statement of course holds true only if we are considering X-ray powder diffraction data; with neutron data, this rotation would need to be considered.

A Z-matrix can be easily adapted to allow for special situations such as disorder. For example, taking the chlorinated example shown earlier, modification of the last lines of the Z-matrix to those shown below allows for the possibility of the rotational disorder of the CH_2Cl group, with occupancies set to 50% in the last column.

```
Cl13  1.7000000  0  109.5000000  0    0.0000000  1   12    1    2 0.5
H14   1.0000000  0  109.5000000  0  120.0000000  0   12    1   13 0.5
H15   1.0000000  0  109.5000000  0  120.0000000  0   12    1   14 0.5
Cl16  1.7000000  0  109.5000000  0    0.0000000  1   12    1    2 0.5
H17   1.0000000  0  109.5000000  0  120.0000000  0   12    1   16 0.5
H18   1.0000000  0  109.5000000  0  120.0000000  0   12    1   17 0.5
```

Sometimes, it can be advantageous to include a dummy atom (D, with zero or near-zero occupancy) in a Z-matrix in order to simplify the definition of a particular problem. For example, consider the case of a centrosymmetric molecule, lying on a crystallographic centre of symmetry, where the molecular centre of symmetry is not coincident with an atom e.g. a benzene ring. A Z-matrix definition of,

```
  D  0.0000000  0    0.0000000  0    0.0000000 0    0    0    0 0.001
  C  1.4000000  0    0.0000000  0    0.0000000 0    1    0    0 1.000
  C  1.4000000  0   60.0000000  0    0.0000000 0    2    1    0 1.000
  C  1.4000000  0  120.0000000  0    0.0000000 0    3    2    1 1.000
```

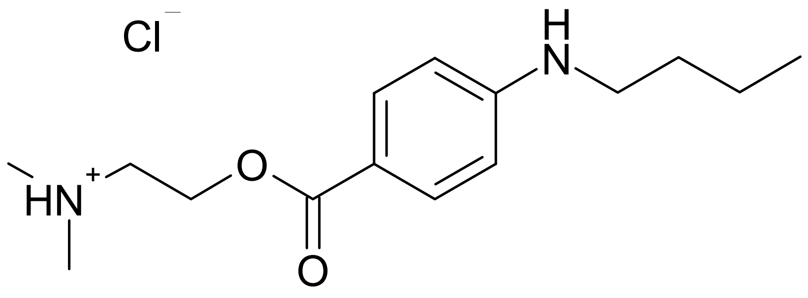
followed by fixing the D atom on the centre of symmetry, gives the required result. Note that for practical reasons in the DASH structure determination package, the dummy atom is set to have a non-zero (but effectively zero) occupancy.

Often, H-atoms can be omitted from the Z-matrix without affecting either the definition or the structure determination. However, in practice, it is easier to assess the correctness of solutions if they are included right from the start of the SDPD process. Their scattering contribution can be ignored in programs such as DASH during the simulated annealing stage so as not to slow down the structure factor calculations. One situation where it is positively advantageous to omit H atoms is in the case of water molecules. If the full molecule is defined, six degrees of freedom are introduced to the problem; if the water is represented by an oxygen atom only, then only positional degrees of freedom need be considered.

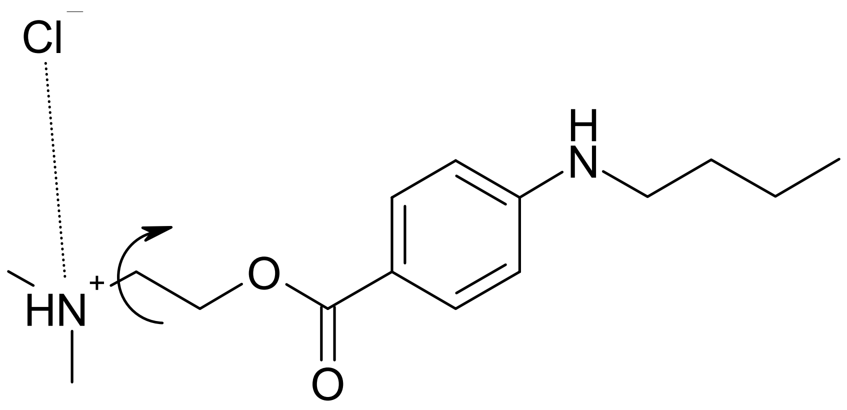
It should now be clear that the Z-matrix acts as a template into which we can plug torsion angle values to generate any desired conformation. Transformation to Cartesian coordinates, followed by reduction to fractional coordinates is then straightforward and translations / rotations within the unit cell can be easily applied to produce a trial crystal structure.

The parameterization of molecules in terms of the unknown torsion angles and external degrees of freedom has proven to be very successful in the context of SDPD. Constraints derived from the crystallography of the system can be applied to these variables; for example, in space group $P1$ the positional degrees of freedom may be fixed at some arbitrary position, or in space group $P2_1$, the y coordinate of the molecule might be fixed at some arbitrary value. Of more interest in the determination of large flexible structures is the use of constraints on torsion angles to significantly reduce the scope of the search problem. The constraints are generally derived from a search of related structures in the Cambridge Structural Database (CSD) and most commonly take the form of bounds on the values that can be adopted by particular torsions. Good examples of such usage can be found in references 2 and 3 and the interested reader is advised to consult them. Direct determination of torsion angles by other methods is also possible [4]. As neither of these methods directly affects the construction of the Z-matrix, they will not be discussed further here.

The concept of applying constraints to *non-bonded* contacts can involve modification of the Z-matrix however. Consider, for example, the case of the chloride ion in tetracaine hydrochloride.



A normal structure determination would involve optimisation of the tetracaine ion position, orientation and conformation, together with the position of the chloride ion. However, a search of the CSD for structures involving the NH^{+}…Cl^{-} motif reveals that the chloride ion does not occupy an arbitrary position relative to the N^{+}; rather, it lies in a somewhat predictable position and in principle, this information can be incorporated into the Z-matrix. It is a straightforward matter to link the tetracaine and chloride fragments into a single Z-matrix where the three positional degrees of freedom of the chloride fragment are exchanged for three different degrees of freedom (virtual bond length, bond angle and bond torsion) that can be optimised with the existing degrees of freedom in the tetracaine molecule.



At first glance, nothing has been gained; we have simply exchanged three variables for three variables. However, these new degrees of freedom can be constrained to chemically sensible values using the prior chemical information obtained from the CSD; for example the $N^{+}\cdots Cl^{-}$ ionic bond distance can be set to 3.06Å with an esd of only 0.05 Å, whilst the N-H···Cl bond angle can be set to 164° with an esd of 10°. In effect, these two variables are removed and the problem collapses to one of determining one more torsion angle within the 'composite molecule'

Automatic generation of Z-matrices of a type suitable for global optimisation presents an interesting algorithmic and programming problem, but that's another story...

1. Leach, A. R. (1996). Molecular Modelling Principles and Applications. Longman.
2. Docherty, A (2004). Crystal Structure Solution and Refinement of Pharmaceutical Molecules Using X-Ray Powder Diffraction. PhD Thesis, University of Strathclyde, Glasgow
3. Nowell, H., Attfield, J.P., Cole, J.C., Cox, P.J., Shankland, K., Maginn, S.J. and Motherwell, W.D.S. *New J. Chem.* 26 (2002) 469-472.
4. Middleton, D.A., Peng, X., Saunders, D., David, W.I.F., Shankland, K and Markvardsen, A.J. *Chem. Commun.* (2002) 1976-1977.

Thanks to Charlie Broder, Tom Griffin and Andrea Johnston for critical comments and helpful suggestions regarding this article.

Including Novel Restraints Supplied by the User to the TNT Refinement Package

Dale. E. Tronrud,
Howard Hughes Medical Institute and Institute of Molecular Biology University of Oregon Eugene, Oregon 97403, USA - Email : dale@uoxray.uoregon.edu ; WWW: http://www.uoxray.uoregon.edu/dale/welcome.html

Abstract

It is not unusual, when refining the structure of a macromolecule, to wish to incorporate some information which was not foreseen by the authors of the software being used. This article discusses the mechanisms in the TNT refinement package which allow this information to be utilized, without modifying the source code for TNT nor recompiling. A program which calculates the data that TNT requires to apply the new restraints must be written by the user.

Introduction

The TNT refinement package (Tronrud et al., 1987) has been available for many years and is still being developed and used for a number of refinement problems. One of its unique features is its ability to accept information from other programs which allows a macromolecular model to be restrained by information not explicitly anticipated within the TNT software.

One example of the successful use of this feature of TNT is the maximum likelihood refinement package Buster (Bricogne, 1997). This package uses the TNT refinement package to perform the calculation for function optimization and stereochemical restraints but replaces TNT's least-squares diffraction residual function with a much more elaborate maximum likelihood target function.

TNT uses the preconditioned conjugate gradient method of function optimization (Tronrud, 1992). This algorithm uses a subset of the Normal matrix (and that subset must be easily invertible) to find an alternative coordinate system for the parameters of the model. The new, abstract, coordinate system is one where the conjugate gradient method (Fletcher & Reeves, 1964) will be more efficient. The conjugate gradient method requires the ability to evaluate the function being optimized given any set of values for the parameters of the model, as well as the gradient of this function. The preconditioned conjugate gradient method requires, in addition, the ability to calculate the diagonal of the Normal matrix.

If the new information being added to the refinement is statistically independent of the restraints implemented in TNT, the expanded restraint function will simply be the sum of the traditional restraints and the function based on the new restraints. Since the derivative operator is a linear operator, the first and second derivatives of the expanded restraint function will, likewise, be the sum of the derivatives of the "traditional" restraint function and the added restraint function. The TNT refinement package can read function values, first derivatives, and the diagonal of second derivatives from a formatted file and add these values to its own corresponding quantities.

To add restraints to a TNT refinement, the user must write a program which will read a TNT coordinate file and produce the value of the restraining function, its first derivative, and diagonal of its second derivative. Proper modification of the TNT refinement script will cause the program to run at the appropriate time, and instruct TNT's program Shift to read the supplied data files in addition to TNT's own.

Choice of Function

TNT is a least-squares refinement package. If the added restraints are to be compatible with the other restraints anticipated by TNT the user should preferably cast them as least-squares restraints, or, failing this, in a negative log-likelihood expression.

The form of a least-squares residual function is

$$f = W \sum_{\text{Observations}} \frac{\left(Q_{\text{obs}} - Q_{\text{calc}}\right)^2}{\sigma^2_{\text{obs}}} \qquad (1)$$

where Q_{obs} is the target value of the restraint, σ_{obs} is the standard deviation of, or the confidence in, the target value, and Q_{calc} is the equivalent quantity calculated from the current model. The weight, W, is used to control the balance between the added restraints and those of TNT. Its value has to be determined empirically by running a series of refinement jobs with a range of values and selecting that value which results in either the lowest free R or all the categories of restraints being met to within the precision of their libraries. If the σ's are accurate the value of the weight will be close to unity.

If the new restraints are cast in the form of a probability, the function will be a form incompatible with a least-squares residual function. This difference, along with the need to maximize probabilities and minimize residuals, makes it impossible to combine within TNT. Taking the negative logarithm of a probability distribution will make the function compatible with a leastsquares residual.

Format of TNT's Data Files

Programs in the TNT refinement package pass data amongst themselves in free-format text files. Each line in one of these files contains a packet of information which can be interpreted independent of its context - The order of the lines in a file is of no importance. The type of data on a line is defined by a keyword written at its head.

To allow TNT to track which piece of information belongs to which part of the function being minimized, the user must choose a name for the information being supplied. You can choose whatever name you wish (with the exception of `RFACTOR`, `GEOMETRY`, and `NCS` which TNT uses) but only the first four letters are significant. In the examples that follow the name "`MINE`" is used.

The keywords used to pass information about restraints to TNT's optimizer program (named Shift) are

`FUNCVAL`	Provides the value of the function being minimized
`DRVC`	Provides the first derivatives of the function with respect to the parameters of a single atom
`CRVC, CRVAC, and CRVBC`	Provides the second derivatives of the function with respect to the parameters of a single atom

Following the keyword is the name chosen for the supplied function.

The `FUNCVAL` statement only needs the function name and the value of the function when evaluated for the current model coordinate file. An example of a `FUNCVAL` statement is

```
FUNCVAL MINE 1.34567E+11
```

A DRVC statement contains the first derivative of the function for the parameters of a particular atom. There is one DRVC statement for each atom in the model. In this statement the keyword is followed by the name chosen by the user, and the derivatives with respect to the five parameters of the atom (X, Y , Z, B, and occupancy). These numbers are followed by the atom's name, the name of the residue containing it, and the name of the chain containing it. An example of a DRVC statement is

```
DRVC MINE 2.12E+02 3.62E+02 1.12E+02 -1.13E+00 7.22E+01 N 1 A
```

These are the five derivatives for the amide nitrogen of residue 1 in chain A.

A question which now must be answered is "Which coordinate system is being used?" TNT was created prior to the establishment of the current "PDB standard" Cartesian coordinate system. While the coordinate system used in TNT to describe the location of atoms is Cartesian, it is not that used in a PDB format coordinate file. To convert from crystallographic "fractional" coordinate system to TNT's Cartesian system use the conversion matrix

$$\mathbf{M}_{\mathbf{c}\to\mathbf{o}} = \begin{pmatrix} 1/a^* & 0 & 0 \\ -a\sin\beta\cos\gamma^* & b\sin\alpha & 0 \\ a\cos\beta & b\cos\alpha & c \end{pmatrix}. \qquad (2)$$

If the residual function does not depend on the placement of the molecule in the crystal then this difference in convention is unimportant. The coordinates found in the TNT coordinate file will already be in an appropriate orthonormal coordinate system.

The diagonal second derivatives are defined with the CRVC, CRVAC, and CRVBC statements. In the most elaborate form available in TNT the diagonal of the second derivative matrix is defined as a symmetric 5x5 block for each atom. (The matrix blocks containing second derivatives for parameters in different atoms are always assumed to be zero in TNT.) The values on the diagonal of this 5x5 block (i.e. $\partial^2 f/\partial X^2$, $\partial^2 f/\partial Y^2$, $\partial^2 f/\partial Z^2$, $\partial^2 f/\partial B^2$, and $\partial^2 f/\partial Occ^2$) are almost always non-zero and significant. The off-diagonal elements of this block are almost always zero (except for $\partial^2 f/\partial B\partial Occ$) and can usually be ignored.

Reflecting this pattern, the CRVC statement contains the values of the five diagonal elements of the block. If all the other elements are equal to zero there is no need to write CRVAC and CRVBC statements. Since the diagonal elements are given on the CRVC statement the order of values on the other statements is a little odd. The values are listed in "threaded order"" which is best described by the figure

$$\begin{pmatrix} 1 & A1 & A5 & B3 & B5 \\ - & 2 & A2 & B1 & B4 \\ - & - & 3 & A3 & B2 \\ - & - & - & 4 & A4 \\ - & - & - & - & 5 \end{pmatrix} \qquad (3)$$

The elements identified as 1, 2, 3, 4, and 5 are given (in that order) on the CRVC statement. The elements identified as A1, A2, A3, A4, and A5 are given on the CRVAC statement and the elements B1, B2, B3, B4, B5 are given on the CRVBC statement. Since the CRVBC statement contains only derivatives which mix positional parameters with the B factor or occupancy it has contained only zeros in our experience. It is unlikely that you will ever write a CRVBC statement.

An example of a CRVC/CRVAC pair is

```
CRVC   MINE 5.90E+02 5.90E+02 642E+02   2.07E-01    8.37E+02 N 1 A
CRVAC  MINE 0.00E+00 0.00E+00 0.00E+00  -1.15E+01   0.00E+00 N 1 A
```

Practical Considerations for Curvatures

The diagonal approximation to the second derivative matrix that is used in TNT is a poor one for many types of restraints. While it works reasonably well for restraints to diffraction amplitudes, geometrical restraints will always have large off-diagonal elements which are a consequence of the "ties" between various sets of atoms.

We have determined, empirically, that when refining with restraints which have large off-diagonal elements (which are always ignored in TNT) the off-diagonal elements in the diagonal 5x5 block should also be ignored. In TNT CRVAC and CRVBC statements are never written for stereochemical and noncrystallographic symmetry restraints.

The diagonal block of second derivatives which is calculated for restraints to diffraction amplitudes for most atoms contains zeros on all off-diagonal elements (except for the $\partial^2 f/\partial B \partial Occ$ element which is not important if the occupancy of the atom is held fixed.) The exception to this pattern occurs when an atom is near a special position in the unit cell. In that situation, the off-diagonal elements of the positional second derivatives are non-zero and are critical to the correct refinement of the location of the atom.

It is recommended that any new restraint programs be written to produce CRVAC and CRVBC statements. Tests comparing the efficiency of the refinement with and without these statements, however, should be run to determine if the presence of this information is helpful or harmful.

Editing the TNT Refinement Script

TNT consists of a number of programs which are coordinated by a script when running refinement. Several csh scripts are supplied for use on unixlike operating systems. The user will have to edit one of these scripts to cause the new program to be executed and its data to be read by TNT.

Regardless of the features of a particular script, all cycles of refinement in TNT have two parts. In the first part (called the "long loop") each helper program calculates the function value, gradient, and diagonal of the second derivative matrix for its restraints.

All this information (along with the previous cycle's shift vector) is read by the program Shift and three things are done:

- A new shift vector is determined by applying the preconditioned conjugate gradient method.
- An estimate of the fraction of this shift vector to be applied is made and written to a temporary file.
- A coordinate file (named shifted.cor) is written which has the trial shift applied.

In the second part of the cycle of refinement (called the "short loops"), the true fraction of the shift vector which optimizes the model is determined. This goal is accomplished by a one-dimensional search along the shift vector. In each iteration, each helper program evaluates its residual function for the trial coordinate file; no derivatives are required here. The program Shift reads the current function values and its own temporary files containing the status of the optimization, and calculates a new estimate. If the new estimate is not significantly different the cycle ends and a new cycle begun.

Refinement of a model against diffraction data and stereochemical restraints is performed by the script `$tntbin/tnt` cycle. The simplest way to add new restraints to a cycle of TNT is to make a copy of this script and edit it. This script runs the programs Rfactor and Geometry (which are TNT's helper programs for restraints on diffraction data and ideal stereochemistry) and then runs Shift. The user should add the commands required to cause their program to calculate its function value and derivatives before the script runs Shift. Then modify the script to cause Shift to read the file containing the appropriate file.

The script commands which run Shift are

```
#
$tntbin/shift << $eof
INCLUDE init.cor
INCLUDE rfactor.dat
INCLUDE geometry.dat
INCLUDE olddir.dat
INCLUDE $control
$eof
if ($status >< 0) exit 1
if (-e tempparm.tmp) rm tempparm.tmp
#
rm rfactor.dat geometry.dat
```

These commands cause Shift to read the starting coordinates (init.cor), the output of the programs Rfactor and Geometry (rfactor.dat and geometry. dat), the previous shift vector (olddir.dat), and the TNT control file (whose name is stored in the symbol $control). If the user's program writes its data in the file mine.dat add the statement

```
INCLUDE mine.dat
```

anywhere in the existing list of INCLUDE statements.

In the second half of the script Rfactor and Geometry are run once again. These programs are followed by Shift. The user will have to add commands to run the added program here (This time only calculating the function value) and add an INCLUDE statement to force Shift to read the user's new file.

Example: Restraining to Torsion Angle Distributions

TNT, as well as other refinement packages (e.g. CNS (Brünger et al., 1998)) restrain molecular models to ideal torsion angles using a simplistic method. Each torsion angle can have several "ideal" values which represent the allowed rotomers for that group of atoms. These ideal values are required to be equally spaced within the circle of 360 degrees, and the width and height of the probability of each of these values are assumed to be identical. Priestle (2003) discusses many of these shortcomings.

To create a restraint on torsion angles which does not have these limitations, one could use the raw probability distributions for the χ angles in place of the distilled list of most probable χ angles usually listed in a rotomer library. Probability distributions suitable for this use are listed in Priestle (2003).

The target function of such a restraint would be

$$f = -W \ln \prod_{j=1}^{n} P_{\chi_j}(\chi_{c,j}(\mathbf{x})), \tag{4}$$

where W is the overall weight of this term relative to the term restraining the model to the diffraction data and the term restraining other stereochemistry items, j is the index of a particular side chain χ angle, $\chi_{c,j}$ is the value of this χ angle calculated from the current model, and $P_{\chi j}$ is the probability distribution of all the values this χ angle could adopt.

The probability of an entire model is the product of the probabilities of each of its χ angles, assuming that the values of these angles are independent of each other. (They are, of course, not independent but to handle the relationship between χ angles requires higher dimensional probability distributions which introduce their own problems. We will continue this example assuming independence.) The negation of the log of this product is calculated to ensure compatibility with TNT.

TNT requires the first, and diagonal of the second derivatives of this function. These derivatives must be with respect to the positional parameters of the atoms in the model. The equation for the first derivatives are

$$\frac{\partial f}{\partial \mathbf{x}_i} = -\sum_{k=}^{\{a,b,\ldots\}} \frac{1}{P_{\chi_k}(\chi_{c,k}(\mathbf{x}))} \frac{dP_{\chi_k}(\chi_{c,k}(\mathbf{x}))}{d\chi_{c,k}(\mathbf{x})} \frac{\partial \chi_{c,k}(\mathbf{x})}{\partial \mathbf{x}_i} \tag{5}$$

and the equation for diagonal of the second derivative matrix is

$$\begin{aligned}
\frac{\partial^2 f}{\partial \mathbf{x}_i^2} = \sum_{k=}^{\{a,b,\ldots\}} \Bigg(& \frac{1}{P^2_{\chi_k}(\chi_{c,k}(\mathbf{x}))} \left(\frac{dP_{\chi_k}(\chi_{c,k}(\mathbf{x}))}{d\chi_{c,k}(\mathbf{x})}\right)^2 \left(\frac{\partial \chi_{c,k}(\mathbf{x})}{\partial \mathbf{x}_i}\right)^2 && (6)\\
& - \frac{1}{P_{\chi_k}(\chi_{c,k}(\mathbf{x}))} \frac{d^2 P_{\chi_k}(\chi_{c,k}(\mathbf{x}))}{d\chi^2_{c,k}(\mathbf{x})} \left(\frac{\partial \chi_{c,k}(\mathbf{x})}{\partial \mathbf{x}_i}\right)^2 && (7)\\
& - \frac{1}{P_{\chi_k}(\chi_{c,k}(\mathbf{x}))} \frac{dP_{\chi_k}(\chi_{c,k}(\mathbf{x}))}{d\chi_{c,k}(\mathbf{x})} \frac{\partial^2 \chi_{c,k}(\mathbf{x})}{\partial \mathbf{x}_i^2} \Bigg) && (8)
\end{aligned}$$

x_i is the positional parameters (i.e. *X, Y* , and *Z*) for the i^{th} atom and {*a, b, . . .*} are the indices of the torsion angles which involve this atom. $\partial\chi_{c,k}(x)/\partial x_i$ indicates how the torsion angle changes as the atom is moved. Values for these derivatives can be found in Tronrud *et al.* (1987).

The derivative of the probability distribution with respect to the χ angle can either be calculated either by discrete differentiation of the histogram itself, or by taking the derivative of a curve fit to the histogram.

The second derivative contains three terms. While the first term is positive in all cases the second and third may be either positive or negative. The optimization method used in TNT requires that the second derivative matrix be positive definite. For a diagonal matrix this means that all the diagonal entries must be positive. The performance of TNT would likely be improved by only calculating the first term.

Further Information

To find further information about the TNT refinement package visit the website http://www.uoxray.uoregon.edu/tnt/

Acknowledgements

This work was supported in part by NIH grant GM20066 to B.W. Matthews.

References

Bricogne, G. (1997). In *Macromolecular Crystallography*, Part A, edited by R. M. Sweet & C. W. Carter, Jr volume 276 of *Methods in Enzymology* pp. 361-423. New York: Academic Press, Inc.

Brünger, A. T., Adams, P. D., Clore, G. M., Gros, P., Grosse-Kunstleve, R. W., Jiang, J.-S., Kuszewski, J., Nilges, M., Pannu, N. S., Read, R. J., Rice, L. M., Simonson, T., & Warren, G. L. (1998). *Acta Cryst*. D54, 905-921.

Fletcher, R. & Reeves, C. (1964). *Computer Journal*, 7, 81-84.

Priestle, J. P. (2003). *J. Appl Cryst*, 36, 34-42.

Tronrud, D. E. (1992). *Acta Cryst*. A48, 912-916.

Tronrud, D. E., Ten Eyck, L. F., & Matthews, B. W. (1987). *Acta Cryst.* A43, 489-501.

Organisation of prior chemical knowledge for macromolecular structure refinement

Alexei A. Vagin and Garib N.Murshudov,
Structural Biology Laboratory, Department of Chemistry, University of York, York, YO10 5YW, UK,
E-mail: garib@ysbl.york.ac.uk and alexei@ysbl.york.ac.uk - WWW: http://www.ysbl.york.ac.uk/~garib/ and http://www.ysbl.york.ac.uk/~alexei/

Abstract

One of the most important aspects in macromolecular structure refinement is the use of prior chemical knowledge. Bond lengths, bond angles and other chemical properties are used in restrained refinement as subsidiary conditions. This contribution describes the organisation of the flexible and human/machine readable dictionary of prior chemical knowledge used by the maximum-likelihood macromolecular refinement program *REFMAC5*. The dictionary stores information about monomers that represent the building blocks of biological macromolecules (amino acids, nucleic acids, and saccharides) and about numerous organic/inorganic compounds commonly found in macromolecular crystallography. It also describes the modifications the building blocks undergo as a result of chemical reactions and the links required for polymer formation. More than 2000 monomer, 100 modification, and 200 link entries are currently available.

1. Introduction

One of the essential components of macromolecular crystal structure refinement is the use of prior information. Prior information available for macromolecular crystallographers can loosely be divided into two families: (a) the available three-dimensional structures of macromolecules deposited within the Protein Data Bank and (b) such invariant chemical properties of macromolecular building blocks as, bond lengths, bond angles, chiral volumes, planes.

A very important, although heavily underused, source of prior information for macromolecular experimental techniques is the PDB (Bernstein *et al*, 1977; Berman *et al*, 2002). It can be expected that many features of the new macromolecular structures are already present within those solved and deposited previously. This aspect of the utilisation of the available information is growing rapidly and there are already some applications of it in such branches of crystal structure analysis as model building (Jones *et al*, 1991) and density modification (Terwilliger, 2003). In future a heavier utilisation of this type of information can be envisaged. A careful analysis and statistically sensible use of this information will definitely enhance and extend the applicability of the currently available experimental techniques for macromolecular structure analysis (e.g. crystallography).

The importance of using known chemical properties, such as bond lengths, bond angles as subsidiary conditions in macromolecular crystallography refinement has been recognised for a long time (Waser, 1963; Jack & Levitt, 1978; Konnert, 1980). The primary justification for the use of these properties is that the experimental data alone are not sufficient to completely define the three-dimensional structure of macromolecules. To extract information from the limited and noisy experimental data it is necessary to use as much as possible chemical information.

This contribution presents the design and the organisation of the dictionary of prior chemical information used by the maximum-likelihood macromolecular crystallographic refinement program *REFMAC5* (Murshudov *et al*, 1997) of the *CCP4* (Collaborative computational project: Number 4, 1994) suite. Since primary purpose of the dictionary is its use by the program *REFMAC5*, in the following sections it is referred as a *REFMAC5* dictionary.

2. Definitions

Monomer. A monomer indicates a chemical unit that, at least formally, can exist independently. For example, amino acids, nucleotides, monosaccharides and ligands are monomers.

Modification. A modification is a formalism that describes the result of changes brought about on a monomer by a chemical reaction. Examples of modification are the N-terminus methylation of amino acids and the methylation of pyranoses at the O1 position.

Link. A link is a formalism that embeds the information required to describe all changes and newly formed bonds occurring when two monomers undergo a chemical reaction that somehow joins them together. Examples of links are *trans/cis* peptide bonds, phosphodiester bonds, and α-4 glycosidic bonds.

Chirality. Chirality is a chemical concept that refers to the property of certain compounds of being non-superimposable to their mirror image. The type of chirality used in the *REFMAC5* dictionary is similar to that used in *SMILES* strings (Weininger, 1988) that is, local chirality as opposed to absolute chirality. Unlike the CIP (Cahn *et al*, 1966) and IUPAC (IUPAC, 1979) conventions for chirality, local chirality is defined only by the immediate surrounding of an atom. Local tetrahedral chirality is the most common one. It is usually present on carbon and nitrogen atoms with sp^3 hybridisation when at least three non-hydrogen atoms are bound to them. Local tetrahedral chirality is defined by its sign. The sign can be either "positive" or "negative". More complex local chiralities are present at metal centres.

Minimal description. Minimal description refers to the minimal information necessary to describe a monomer uniquely. It consists of the monomer name, the list of its atoms identification codes and symbols, its bonds list and orders, and optionally the chemical group to which it belongs to (peptide, pyranose, etc.). If required, the configuration of the monomer can be defined using chiralities.

Complete description. Complete description is a monomer description that contains all information about its internal chemical structure. In addition to the items present in the minimal description it also contains a tree representation of the monomer as well as its bond lengths, bond angles, torsion angles. When required, planes and chiral centres are also defined. For the appropriate parameters standard deviations are given.

3. Dictionary of prior chemical information

The dictionary used by the program *REFMAC5* has been designed according flexibility criteria. It is largely based on the monomer approach and allows dynamic definition of links and modifications. It contains carefully analysed descriptions for most common monomers, modifications and links.

REFMAC5 dictionary is written in an extended mmCIF format (Bourne *et al*, 1997). This is based on the star style (Hall, 1991) and the CIF format (Hall *et al*, 1991) used in small molecules crystallography. The attractive side of the mmCIF format is that any data file based on it can easily be extended without affecting the functionalities of programs already using it.

3.1 General organisation and current state of the dictionary

REFMAC5 dictionary contains a list of monomers, modifications, and links along with their descriptions. Monomer descriptions define the internal coordinate of independent compounds. Modifications and links encapsulate the changes brought about on them by chemical reactions. Modifications typically act on a single monomer whilst links join monomers together.

The currently distributed version of the dictionary has entries for all amino acids as well as for many of their possible modifications, for all nucleic acids and for some of their modifications, and for most common sugars and their modifications. It has also entries for many organic and inorganic compounds frequently encountered when solving macromolecular structures. As some monomers have several well established common names the dictionary contains a list of synonyms capable of handling them.The dictionary also contains frequently encountered links like *trans/cis* and methylated peptide links, sugar-sugar and sugar-protein links, as well as DNA/RNA links.

Current version of the dictionary contains more than 2000 monomer, 100 modification, and 200 links. Such a large dictionary covers most common users' needs. A full list of monomers, modifications and links available within the *REFMAC5* dictionary can be found at the web-page http://www.ysbl.york.ac.uk/~alexei/dictionary.html.

The dictionary can easily be extended by users. Users can create and organise personal monomer entries as well as modifications and links. In case of conflict user's definitions always override that stored within the distributed dictionary.

At present, the dictionary is used mainly by the program *REFMAC5* for restrained refinement. However, its organisation is so that other programs dealing with macromolecules can use it. For example, the model building program *COOT* (Emsley and Cowtan, 2004) employs it. Applications for molecular simulation and modelling that use the *REFMAC5* dictionary are currently being developed.

3.2 Monomers

For a monomer to be completely defined information must be available about its constituting atom(s) and, if present, about its bonds, angles, torsion angles, planes, and chiral centres. Examples of complete monomer descriptions are given in Table 1.

Table 1a. Example of complete monomer description.

This example shows the complete monomer description of the pyranose β-D-glucose. For compactness, most description categories given contain only a representative set of items. Missing items are represented by "..." symbols. The first category (*_chem_comp*) is the general category. It contains the name of the monomer alongside with its long name and the name of the group to which it belongs to. In this category there is an indication of the level of monomer description. If this item has the value "M" the entry has a minimal description. In this case the value is "." which indicates a complete description. The second category (*_chem_comp_atom*) describes atoms with their names, element names, atom types, and atom charges. It can also contain a monomer representation in Cartesian coordinates. The third category (*_chem_comp_tree*) is the acyclic graph description. Additional bonds are present to indicate ring enclosure also. These bonds have the label "ADD". Beginning and end of the tree are labelled with "START" and "END", respectively. The fourth category (*_chem_comp_bond*) lists bonds together with their bond lengths, bond orders and uncertainties. Other categories present are for bond angles (*_chem_comp_angle*), torsion angles (*_chem_comp_tor*), and chiralities (*_chem_comp_chir*). When required, planes are indicated by the category (*_chem_comp_plane_atom*). The latter category is not present in this example as β-D-glucose does not need planarity restraints.

```
#
data_comp_list
loop_
_chem_comp.id
_chem_comp.three_letter_code
_chem_comp.name
_chem_comp.group
_chem_comp.number_atoms_all
_chem_comp.number_atoms_nh
_chem_comp.desc_level
GLC-b-D  GLC 'beta_D_glucose                          ' D-pyranose         24  12
.
#
data_comp_GLC-b-D
#
loop_
_chem_comp_atom.comp_id
_chem_comp_atom.atom_id
_chem_comp_atom.type_symbol
_chem_comp_atom.type_energy
_chem_comp_atom.partial_charge
 GLC-b-D       C1     C    CH1       0.000
 GLC-b-D       H1     H    HCH1      0.000
 ...
 ...
 GLC-b-D       HO6    H    HOH1      0.000
 GLC-b-D       O5     O    O2        0.000
loop_
_chem_comp_tree.comp_id
_chem_comp_tree.atom_id
_chem_comp_tree.atom_back
_chem_comp_tree.atom_forward
_chem_comp_tree.connect_type
 GLC-b-D  C1     n/a    C2     START
 GLC-b-D  H1     C1     .      .
 ...
 ...
 GLC-b-D  O6     C6     HO6    .
 GLC-b-D  HO6    O6     .      .
 GLC-b-D  O5     C5     .      END
 GLC-b-D  O5     C1     .    ADD
loop_
_chem_comp_bond.comp_id
_chem_comp_bond.atom_id_1
_chem_comp_bond.atom_id_2
_chem_comp_bond.type
_chem_comp_bond.value_dist
_chem_comp_bond.value_dist_esd
 GLC-b-D  O1     C1        single       1.410    0.020
 GLC-b-D  C2     C1        single       1.524    0.020
 ...
 ...
 GLC-b-D  HO6    O6        single       0.980    0.020
 GLC-b-D  C1     O5        single       1.410    0.020

loop_
_chem_comp_angle.comp_id
_chem_comp_angle.atom_id_1
_chem_comp_angle.atom_id_2
_chem_comp_angle.atom_id_3
_chem_comp_angle.value_angle
_chem_comp_angle.value_angle_esd
 GLC-b-D  H1     C1     O1      109.470    3.000
 GLC-b-D  O1     C1     C2      109.470    3.000
 ...
 ...
 GLC-b-D  C6     O6     HO6     109.470    3.000
 GLC-b-D  C5     O5     C1      111.800    3.000
loop_
_chem_comp_tor.comp_id
_chem_comp_tor.id
_chem_comp_tor.atom_id_1
_chem_comp_tor.atom_id_2
_chem_comp_tor.atom_id_3
_chem_comp_tor.atom_id_4
_chem_comp_tor.value_angle
_chem_comp_tor.value_angle_esd
_chem_comp_tor.period
 GLC-b-D  var_1    C1     C2     O2     HO2        0.000   20.000   1
 GLC-b-D  var_2    C1     C2     C3     C4       -50.095   20.000   3
 ...
 ...
 GLC-b-D  var_11   C5     O5     C1     C2       -55.889   20.000   3
 GLC-b-D  var_12   O5     C1     C2     C3        55.889   20.000   3
loop_
_chem_comp_chir.comp_id
_chem_comp_chir.id
_chem_comp_chir.atom_id_centre
_chem_comp_chir.atom_id_1
_chem_comp_chir.atom_id_2
_chem_comp_chir.atom_id_3
_chem_comp_chir.volume_sign
 GLC-b-D  chir_01  C5     C4     O5     C6         positiv
 GLC-b-D  chir_02  C4     C3     O4     C5         positiv
 GLC-b-D  chir_03  C3     C2     O3     C4         negativ
 GLC-b-D  chir_04  C2     C1     O2     C3         positiv
 GLC-b-D  chir_05  C1     O1     O5     C2         positiv
```

Table 1b. Example of the complete monomer description with metal chirality.

Description of seven coordinated calcium. Chirality sign is cross5 shown that there are 5+2=7 atoms surrounding the central atom. After the centre, chirality definition has information about up to eight atoms. First atom indicated by *_chem_comp_chir.atom_id_1* shows starting atom, second atom indicated by *_chem_comp_chir.atom_id_2* shows the end and all others show surrounding atoms (see Figure 2). Apart from centre all atoms are optional. It allows flexible definition of distorted coordination. However there must be at least two atoms surrounding the central atom.

```
global_
_lib_name         mon_lib
_lib_version      4.3
_lib_update       11/06/03
# ------------------------------------------------
#
# ---   LIST OF MONOMERS ---
#
data_comp_list
loop_
_chem_comp.id
_chem_comp.three_letter_code
_chem_comp.name
_chem_comp.group
_chem_comp.number_atoms_all
_chem_comp.number_atoms_nh
_chem_comp.desc_level
OC7      .   'CALCIUM ION, 7 WATERS COORDINATED   ' non-polymer    22   8 .
#
```

```
# --- DESCRIPTION OF MONOMERS ---
#
global_
_lib_name         mon_lib
_lib_version      4.3
_lib_update       11/06/03
# ------------------------------------------------
#
# ---   LIST OF MONOMERS ---
#
data_comp_list
loop_
_chem_comp.id
_chem_comp.three_letter_code
_chem_comp.name
_chem_comp.group
_chem_comp.number_atoms_all
_chem_comp.number_atoms_nh
_chem_comp.desc_level
OC7 . 'CALCIUM ION, 7 WATERS COORDINATED' non-polymer   22   8 .
#
# --- DESCRIPTION OF MONOMERS ---
#
data_comp_OC7
#
loop_
_chem_comp_atom.comp_id
_chem_comp_atom.atom_id
_chem_comp_atom.type_symbol
_chem_comp_atom.type_energy
_chem_comp_atom.partial_charge
_chem_comp_atom.x
_chem_comp_atom.y
_chem_comp_atom.z
 OC7    O7    O  O    0.000   0.000  0.001  0.000
 .......
 OC7    HO12  H  H    0.000   0.241  4.292  0.001
 OC7    HO11  H  H    0.000  -1.011  3.040  0.002
loop_
_chem_comp_bond.comp_id
_chem_comp_bond.atom_id_1
_chem_comp_bond.atom_id_2
_chem_comp_bond.type
_chem_comp_bond.value_dist
_chem_comp_bond.value_dist_esd
 OC7    O1    CA   single   2.320  0.020
 ......
 OC7    HO72  O7   single   1.040  0.020
loop_
_chem_comp_angle.comp_id
_chem_comp_angle.atom_id_1
_chem_comp_angle.atom_id_2
_chem_comp_angle.atom_id_3
_chem_comp_angle.value_angle
_chem_comp_angle.value_angle_esd

 OC7   O7  CA   O5   180.000  3.000
 .....
 OC7   O2  CA   O5    90.000  3.000
```

```
loop_
_chem_comp_chir.comp_id
_chem_comp_chir.id
_chem_comp_chir.atom_id_centre
_chem_comp_chir.atom_id_1
_chem_comp_chir.atom_id_2
_chem_comp_chir.atom_id_3
_chem_comp_chir.volume_sign
_chem_comp_chir.atom_id_4
_chem_comp_chir.atom_id_5
_chem_comp_chir.atom_id_6
_chem_comp_chir.atom_id_7
_chem_comp_chir.atom_id_8
 OC7   chir_01  CA   O5   O7  O1   cross5
                         O4    O2   O3  O6      .
```

Monomers are described by the following categories:

General category. This category contains the short and full monomer names, the monomer three letter PDB code and the group it belongs to (peptide, DNA/RNA, pyranose, non-polymer). Group names are an important part of the monomer description as they facilitate monomer handling. For example, if the monomer belongs to the group called "peptide" then all links and modifications described for peptides can be applied to it. Moreover, the group type defines whether a monomer can belong to a chain (polypeptide, DNA/RNA or polysaccharide chains).

Atom category. This category lists atom and element names and their chemical types and charges. It may also contain Cartesian coordinates.

Tree category. This category describes the mathematical tree (acyclic graph) corresponding to the monomer chemical connectivity. It is used to generate coordinates. Missing atoms, e.g. hydrogens, are restored using this tree.

Bond category. This category contains the list of bonded atoms, bond types, and ideal values of bond lengths and uncertainties associated with them. Alongside with the atom category this category defines completely the chemical structure of the monomer. In mathematical terms such a structure is called a coloured graph. Edges are coloured by bond orders and vertices are coloured by chemical types.

Angle category. This category contains the three-atoms list of all possible angles in the monomer as well as their ideal values and associated uncertainties.

Torsion angle category. This category contains the four-atoms list of torsion angles, their types and names, their ideal values and associated uncertainties, and their period. The latter value represents the number of energetic minima along the torsion angle. For example, χ angles along the C_α-C_β bond of glutamine residue have a period equal to three. A torsion angle can be constant or variable. Constant torsion angles generally involve atoms belonging to the same plane or atoms along double bonds. Usually, these torsion angles have period equal to zero or one as they can have a single value only.

Plane category. This category contains the list of planes and of all atoms belonging to them.

Chirality category: tetrahedral chirality. This category contains the list of all chiral centres. For each chiral centre it also lists the central atom, the atoms bonded to it and the sign of the chiral volume. The current version of the dictionary allows undefined signs using "both" or "anomer" keywords. If the keyword "both" is used the chirality of the monomer can change during restrained refinement. If the keyword "anomer" is used the chirality is fixed and its sign is defined by the input coordinates. If for a monomer in a crystal there are two or more configurations all of them can be simultaneously handled during refinement by assigning the keyword “anomer” to each chiral centre.

Chirality category: Metal chirality. It is a special case of the general *chirality* category. This type of chirality allows description of surrounding of metals. Keyword used for this is “cross*n*” where *n* is between zero and six (Table 1b and Figure 1).

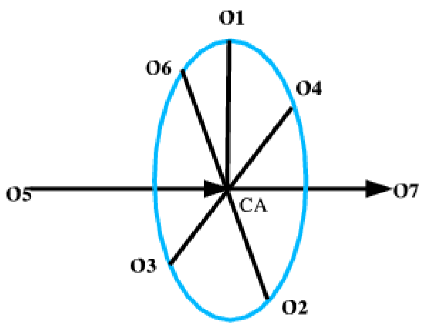


Figure 1: *Schematic view of metal chirality*

Ideal values for bond lengths and bond angles for standard amino acids present in the dictionary have been taken from Engh and Huber (1991). Ideal values for bond lengths and angles for nucleic acids have been taken from (Kennard and Taylor, 1982). Ideal values for bond lengths and angles for most saccharides have been taken from Sanger (1983).

At present, about 1000 monomers out of the 2000 available in the *REFMAC5* dictionary are present with a complete description. The remaining ones are present with a minimal description. Work is in progress to deliver in the shortest time possible a dictionary in which all entries are present with checked complete descriptions.

3.3 Modifications

A modification is a formalism that describes changes brought about on a single monomer by chemical reactions. An example of modification is shown Figure 2a. Its dictionary description is given in Table 2. A modification allows atoms, bonds, angles, torsion angles, planes, chiral centres to be added to or deleted from monomers. The use of modifications greatly reduces the number of monomer descriptions that need to be stored and allows describe properly links between monomers as some of them require monomers to first undergo modifications prior linkage. Modifications can also be used for non-chemical changes on monomers such as changes in residues name. This is a convenient way of handling cases of multiple monomer names. In such cases the modification keyword is “RENAME”. This keyword is also used to overcome the three-letter restriction imposed by the PDB convention.

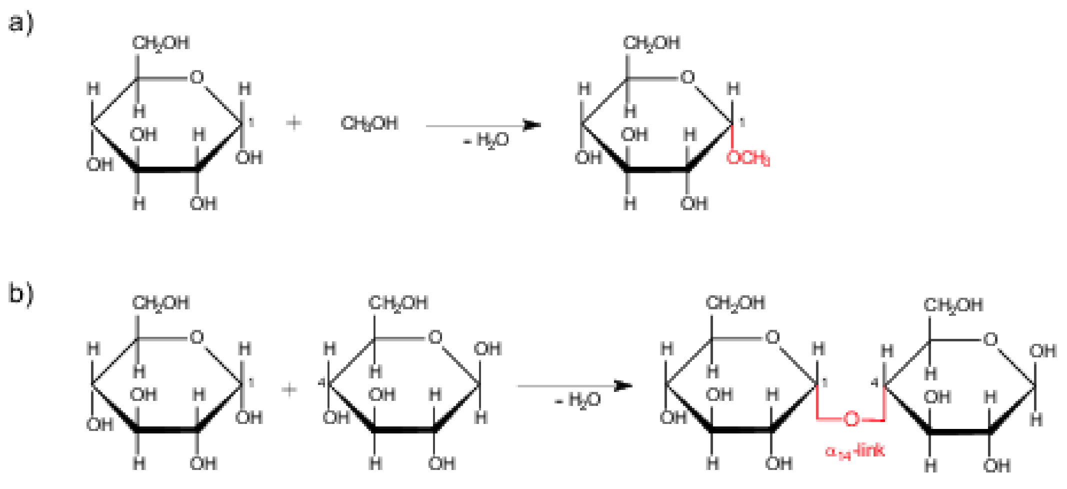


Figure 2: *a) Example of a sugar modification. The condensation of α-D-glucose with methanol gives methyl-α-D-glucoside; (b) Example of sugar link. The disaccharide β-maltose is formed by condensation of α-D-glucose with β-D-glucose. The glycosidic bond is an α-4 link.*

Table 2. Example of modification.

This example describes the methylation at the O1 position of pyranoses. See Figure 1a) for a graphical representation of this modification. The first - general - category (*_chem_mod*) reports the code for the modification "O1MET" and describes whether the modification is to be applied to a only a particular monomer or to group of monomers. "O1MET" modification can be applied to all monomers belonging to "pyranose" group. The (*_chem_mod_atom*) category describes the list of all added, deleted or changed atoms. The following category (*_chem_mod_bond*) describes all added or deleted bonds. In a similar manner the tree structure (*_chem_mod_tree*), bond angles (*_chem_mod_angle*), torsion angles (*_chem_mod_tor*), planes (*_chem_mod_plane_atom*), and chiralities (*_chem_mod_chir*) when affected by the modification are handled.

```
data_mod_list
loop_
_chem_mod.id
_chem_mod.name
_chem_mod.comp_id
_chem_mod.group_id
O1MET  O1_metyl_of_sugar   .  pyranose

data_mod_O1MET

loop_
_chem_mod_atom.mod_id
_chem_mod_atom.function
_chem_mod_atom.atom_id
_chem_mod_atom.new_atom_id
_chem_mod_atom.new_type_symbol
_chem_mod_atom.new_type_energy
_chem_mod_atom.new_partial_charge
 O1MET    change   O1      .      .    O2      0.000
 O1MET    delete   HO1     .      .    .       0.000
 O1MET    add      .       HM3    H    HCH     0.000

loop_
_chem_mod_bond.mod_id
_chem_mod_bond.function
_chem_mod_bond.atom_id_1
_chem_mod_bond.atom_id_2
_chem_mod_bond.new_type
_chem_mod_bond.new_value_dist
_chem_mod_bond.new_value_dist_esd
 O1MET    add      O1      CM        single      1.420    0.020

 O1MET    add      CM      HM3       single      0.960    0.020
loop_
_chem_mod_angle.mod_id
_chem_mod_angle.function
_chem_mod_angle.atom_id_1
_chem_mod_angle.atom_id_2
_chem_mod_angle.atom_id_3
_chem_mod_angle.new_value_angle
_chem_mod_angle.new_value_angle_esd
 O1MET    add      C1      O1      CM      120.000    3.000
```

3.4 Links

The link formalism allows join monomers together. An example of link is shown Figure 2b. Its description is given in Table 3. Links can be considered the external counterpart of monomer descriptions. Whereas monomer descriptions give the internal structure of single chemical compounds link descriptions define in detail the result of chemical reactions between monomers. Link descriptions contain information about the monomers or the group of monomers they act on as well as about the modifications these monomers should undergo prior linkage. In the current version of the dictionary a link can form only one bond. However, the introduction of several angles, torsion angles, planes, chiral centres is allowed.

Table 3. Example of link.

This example describes the α1-4 pyranose link. See Figure 1b) for a graphical representation of this link. The first - general - category *_chem_link* describes the name, the link identification code, and the scope of this link. It also contains pointers to the modifications the monomers should undergo before the link can be applied. This link requires that the monomers belong to "pyranose" group and that the first and second monomers undergo DEL-HO4 and DEL-O1 modifications, respectively. The description for both these modification need to be available before the link can be applied. Other categories give the list of bonds (*_chem_link_bond*), bond angles (*_chem_link_angle*), torsion angles (*_chem_link_tor*), chiralities (*_chem_link_chir*), etc. with their "ideal" values. Atom names in the link description are always given together with the monomer numbers they belong to.

```
#
data_link_list
loop_
_chem_link.id
_chem_link.comp_id_1
_chem_link.mod_id_1
_chem_link.group_comp_1
_chem_link.comp_id_2
_chem_link.mod_id_2
_chem_link.group_comp_2
_chem_link.name
ALPHA1-4 .        DEL-HO4  pyranose .        DEL-O1   pyranose
glycosidic_bond_alpha1-4

data_link_ALPHA1-4
#
loop_
_chem_link_bond.link_id
_chem_link_bond.atom_1_comp_id
_chem_link_bond.atom_id_1
_chem_link_bond.atom_2_comp_id
_chem_link_bond.atom_id_2
_chem_link_bond.type
_chem_link_bond.value_dist
_chem_link_bond.value_dist_esd
 ALPHA1-4 1 O4      2 C1        single      1.439    0.020
loop_
_chem_link_angle.link_id
_chem_link_angle.atom_1_comp_id
_chem_link_angle.atom_id_1
_chem_link_angle.atom_2_comp_id
_chem_link_angle.atom_id_2
_chem_link_angle.atom_3_comp_id
_chem_link_angle.atom_id_3
_chem_link_angle.value_angle
_chem_link_angle.value_angle_esd
 ALPHA1-4 1 C4      1 O4      2 C1      108.700    3.000
 ALPHA1-4 1 O4      2 C1      2 O5      112.300    3.000
 ALPHA1-4 1 O4      2 C1      2 C2      109.470    3.000
 ALPHA1-4 1 O4      2 C1      2 H1      109.470    3.000
loop_
_chem_link_tor.link_id
_chem_link_tor.id
_chem_link_tor.atom_1_comp_id
_chem_link_tor.atom_id_1
_chem_link_tor.atom_2_comp_id
_chem_link_tor.atom_id_2
_chem_link_tor.atom_3_comp_id
_chem_link_tor.atom_id_3
_chem_link_tor.atom_4_comp_id
_chem_link_tor.atom_id_4
_chem_link_tor.value_angle
_chem_link_tor.value_angle_esd
_chem_link_tor.period
 ALPHA1-4 ALPHA_1  1 O4     2 C1     2 C2     2 C3        0.00  20.0 1
 ALPHA1-4 ALPHA_2  1 C4     1 O4     2 C1     2 C2        0.00  20.0 1
 ALPHA1-4 ALPHA_3  1 C3     1 C4     1 O4     2 C1        0.00  20.0 1
loop_
_chem_link_chir.link_id
_chem_link_chir.atom_centre_comp_id
_chem_link_chir.atom_id_centre
_chem_link_chir.atom_1_comp_id
_chem_link_chir.atom_id_1
_chem_link_chir.atom_2_comp_id
_chem_link_chir.atom_id_2
_chem_link_chir.atom_3_comp_id
_chem_link_chir.atom_id_3
_chem_link_chir.volume_sign
 ALPHA1-4   2 C1      1 O4      2 O5      2 C2      negativ
```

3.5 Atom types library

Although the *REFMAC5* dictionary is largely based on monomers it also contains an atom type library. At present, it contains about 300 atom types. It includes all chemical elements as well as many atom types commonly encountered in chemistry. Each entry has information about the chemical element the atom type belongs to as well as about its van der Waals (VDW) and ionic radii. The atom type library contains also information about possible bonds between atom types. For many pairs of atom types bond orders and bond lengths are tabulated. Angles corresponding to some of the atom type triplets are also listed. The atom types library is in mmCIF format. Therefore, it can easily be updated and extended. A full list of all atom types library entries can be found at the web-page http://www.ysbl.york.ac.uk/~alexei/dictionary.html.

The bond lengths listed in the atom types library have been taken from the International Table for Crystallography (Allen *et al* 1992; Orpen *et al*, 1992. VDW and ionic radii of atoms have been taken from various sources including Greenwood and Earnshaw (1989) and Cotton and Wilkinson (1972). Unfortunately, to our knowledge there is no single general reference for bond angles. Some of the angles have been taken from the examples of the Cambridge Structural Database (Allen, 2002), others have been derived using general information about atoms i.e. their hybridisation and the nature of the surrounding atoms.

The atom types library serves two main purposes: a) it provides information about VDW and ionic radii as well as about atoms' hydrogen bonding capability that is used to define non-bonding interactions in the course of refinement; b) it provides information about initial bond lengths and angles when new monomer entries are created.

4 Web resources

Two web resources can produce complete monomer descriptions compatible with *REFMAC5*. The first resource is hosted at the European Bioinformatics Institute (Golovin *et al*, 2004) and can be accessed from the web address http://www.ebi.ac.uk/msd-srv/chempdb/cgi-bin/cgi.pl. The second one is the program *PRODRG* (Aalten *et al*, 1996) which can be found at the web address http://davapc1.bioch.dundee.ac.uk/programs/prodrg/prodrg.html.
Other sources like the *CORINA* suite (Sadowski *et al*, 1994) available from http://www2.chemie.uni-erlangen.de/software/corina/index.html give coordinates that can be used to create *REFMAC5* monomer descriptions. *CORINA* can be used with help of the *CACTVS* (Ihlenfeldt *et al*, 1994) interface.
There are several databases that can produce coordinates for various sugars. These include:

http://www.cermav.cnrs.fr/databank/mono/index2.html - monosaccharide database
http://www.cermav.cnrs.fr/databank/disacch/index.html - disaccharide database,
http://www.dkfz-heidelberg.de/spec/sweet2/doc/index.php - sugar database.

One should be careful when using various databases. Most databases use such built-in chemical assumptions as protonation of carboxyl oxygens. These assumptions can affect geometric parameters.

5 Conclusions and future perspectives

A flexible, machine/human readable dictionary of monomers, links, modification and related items has been created and tested on a wide range of compounds. The dictionary is currently used for macromolecular restrained refinement by the program *REFMAC5*. It can also be used by other macromolecular programs like model building and macromolecular modelling and simulations applications.

Flexibility in the organisation of the dictionary allows researchers to add personal entries and to override existing descriptions. The most common crystallographic restraints are dealt with in an automatic manner. Complicated cases can also be handled with some user intervention.

The dictionary is distributed by *CCP4* under the Part 0 licence that is LGPL compatible. Programs and interface are available from *CCP4* under the Part 2 licence. Neither programs nor dictionary nor algorithms have been patented to make sure that they are available to community of users as well as developers.

For further information about *REFMAC5* dictionary and tools to create new dictionary entries see: (Vagin *et al*, 2004) and `http://www.ysbl.york.ac.uk/~alexei/dictionary.html`.
Information about the latest version of the program REFMAC5 and its dictionary can be found in: `http://www.ysbl.york.ac.uk/~refmac/index.html`

This work was supported by grants from the Wellcome Trust (GNM), BBSRC (AAV). We thank people from the YSBL especially Roberto Steiner, Andrey Lebedev, Fei Long, Liz Potterton and Stuart McNicholas, *CCP4* staff and the user community for their continuous support and encouragements.

References

van Aalten, D., Bywater, R., Findlay, J., Hendlich, M., Hooft, R. and Vriend, G. (1996). *Journal of Computer Aided Molecular Design*, **10**, 255--262.

Allen, F. (2002). *Acta Cryst.* **B58**, 380--388.

Allen, F., Kennard, O., Watson, D., Brammer, L., Orpen, A. and Taylor, R. (1992). In *International Tables for Crystallographers*, edited by A. Wilson, **vol. C**, pp. 685-706. Dordrech, Boston, London: Kluwer Academic Publishers.

Berman, H.M., Battistuz, T., Bhat, T.N., Bluhm, W.F., Bourne, P.E., Burkhardt, K., Feng, Z., Gilliland, G.L., Iype, L., Jain, S.,

Fagan, P., Marvin, J., Padilla, D., Ravichandran, V., Schneider, B., Thanki, N., Weissig, H., Westbrook, J.D. and Zardecki, C. (2002). *Acta Cryst.* **D58 6 Part 1**, 899--907.

Bernstein, F.C., Koetzle, T.F., Williams, G.J., Meyer Jr, E.F., Brice, M.D., Rodgers, J.R., Kennard, O., Shimanouchi, T. and Tasumi, M. (1977). *J. Mol. Biol.* **112(3),** 535--542.

Bourne, P., Berman, H., McMahon, B., Watenpaugh, K., Westbrook, J. and Fitzgerald, P. (1997). *Meth. Enzymol.* **277**, 571-590.

Cahn, R., Ingold, C. Prelog, V. (1966). *Angew. Chem.* **78**, 413-447.

Collaborative Computational Project Number 4. (1994). *Acta Cryst.* **D50**, 760-763.

Cotton, F. and Wilkinson, G. (1972). *Advanced Inorganic Chemistry*. Interscience Publishers.

Emsley, P. and Cowtan, K. (2004). *Acta Cryst.* **Section D**, in press.

Engh, R.A. and Huber, R. (1991). *Acta Cryst.*} **A47**, 39--400.

Golovin, A., Oldfield, T., Tate, J., Velankar, S., Barton, G., Boutselakis, H., Dimitropoulos, D., Fillon, J., Hussain, A., Ionides, J., John, M., Keller, P., Krissinel, E., McNeil, P., Naim, A., Newman, R., Pajon, A., Pineda, J., Richedi, A., Copeland, J., Sitnov, A., Sobhany, S., Suarez-Uruena, A., Swiminathan, J., Tagari, M., Tromm, S., Vranken, W. and Henrick, K. (2004). *Nucleic Acid Research*, **32**, D211-D216.

Greenwood, N. and Earnshaw, A. (1989). *Chemistry of the Elements*. Pergamen Press.

Hall, S. (1991) *J. Chem. Inf. Comp. Sci*, **31**, 326--333.

Hall, S., Allen, A. and Brown, I. (1991). *Acta Cryst.*, A47, 655-685.

Ihlenfeldt, W., Takahashi, Y., Abe, H. and Sasaki, S. (1994). *J.Chem. Inf. Comp. Sci.* **34**, 109--116.

IUPAC (1979). *Nomenclature of Organic Chemistry*, Sections A, B, C, D, E, F, and H. Pergamon Press.

Jack, A. and Levitt, M. (1978). *Acta Cryst.* **A34**, 931--935.

Jones, A. T. (1991). *Acta Cryst.* **D47**, 442-450.

Kennard, O. and Taylor, R. (1982). *J. Am. Chem. Soc.* **104**, 3209-3212.

Konnert, J. and Hendrickson, W. (1980). *Acta Cryst.*, *A36*, 344--350.

Murshudov, G.N., Vagin, A.A. and Dodson, E.J. (1997). *Acta Cryst.*, **D53**, 240--255.

Orpen, A., Brammer, L., Allen, F., Kennard, O., Watson, D. and Taylor, R. (1992). *International Tables for Crystallographers*, edited by A. Wilson, **vol.C**, pp. 707--791. Dordrech, Boston, London: Kluwer Academic Publishers.

Sadowski, J., Gasteiger, J. and Klebe, G. (1994). *Chem. Inf. Comput. Sci*, **34**, 1000-1008.

Saenger, W. (1983). *Principles of Nucleic Acid Structure*. Springer-Verlag.

Terwilliger, T. (2003). *Acta Cryst.*, **D59**, 1688--1701.

Vagin, A., Steiner, R., Lebedev, A., Potterton, L., McNicholas, S., Long, F and Murshudov, G.N. (2004). *Acta Cryst.* **Section D** in press.

Waser, J. (1963). *Acta Cryst.* **16**, 1091--1094.

Weininger, D. (1988). J. *Chem. Inf. Comput. Sci.*, **28**, 31--.

X-rays don't see atoms

David Watkin,
Chemical Crystallography, Department of Chemistry, University of Oxford, Chemistry Research Laboratory, Mansfield Road, Oxford, OX1 3TA, UK - Email : david.watkin@chem.ox.ac.uk ; WWW: http://www.xtl.ox.ac.uk/ and http://www.chem.ox.ac.uk/researchguide/djwatkin.html

In addition to the three dimensional Bragg equation (1), there are three more fundamental equations needed to sum up modern X-ray structure analysis.

$$4\,\mathrm{Sin}^2\theta/\lambda^2 = h^2\cdot a^{*2}....+2\,klb^*c^*\,\mathrm{Cos}\,\alpha^* \qquad \mathbf{1}$$

The first explains what happens during the X-ray diffraction experiment, in which the incident wave front falls upon a periodically repeating pattern of varying electron density (2). Note that 'atoms' do not come into this equation. From the e^i term, we can see that the diffracted beams have both magnitude and a phase shift with respect to the un-diffracted wave front.

$$F_{hkl} = \int\int\int \rho_{xyz}\cdot e^{2\pi i(hx+ky+lz)}\,\partial x.\partial y.\partial z \qquad \mathbf{2}$$

The second shows that if one has measures of the diffracted magnitudes and phase angles, these can be used to compute the value of the electron density at any (and every) point within the crystallographic unit cell (3). Most diffraction experiments yield good estimates of the diffracted intensities, but phase angles are much more difficult to measure, and in practical X-ray structure determination these phases are not measured. Instead, values can be estimated from the intensities themselves by a process called ‘Direct Methods’ or by other methods.

$$\rho_{xyz} = \frac{1}{v}\sum\sum\sum |F|_{hkl}\, e^{-2\pi i(hx+ky+lz-\alpha_{hkl})} \qquad \mathbf{3}$$

Equations 2 and 3 are reasonably clear-cut and offer a well defined view of the physics of the experiment, provided the periodically varying electron density in the crystal *is* periodic on the time scale of the experiment.

The third equation requires a much greater leap of confidence, yet is scarcely ever questioned by chemists using crystallography as an analytical tool (4). This equation is related to equation 2, except that the integration over a continuously varying periodic electron density has been replaced by a summation over a periodic array of atoms. The popularity of this model undoubtedly comes from the fact that it provides a very efficient representation of the electron distribution in the sample[1], and that experience has shown that this approximation serves well for the computation of other physical properties of materials.

$$F_{hkl} \approx \sum f_j\cdot e^{2\pi i(hx_j+ky_j+lz_j)} \qquad \mathbf{4}$$

If it were practical to measure the phase angle of every diffracted beam in the same time that it takes to measure its intensity, then computation of the continuous electron density would provide a real (time and space averaged) image of the material, which could be used to assess the appropriateness of an ‘atomic’ model. Since this cannot be done, we resort to using improved atomic models to provide (through equation 4) improved estimates of the phases to be used in combination with the observed

[1] A 10-atom unit cell in P1 requires 40 parameters in the isotropic adp representation, but would require five and a half thousand electron density values for a three dimensional grid sampled at 1/3 of an Angstrom.

amplitudes for the creation of improved electron density maps. The improvement in the atomic model can come from examining the computed electron density for new features, or by mathematically adjusting the actual atomic parameter values to improve the agreement between the observed and calculated structure amplitudes, usually by some kind of lest-squares method. There is a tacit belief that a model which gives good agreement between the observed and computed amplitudes is a good model, and hence yields good estimates of the phase angles. These two procedures have now become so well automated that it is rare that a structure analyst (at least in small molecule crystallography) will actually look at electron density maps.

Luckily this tidy view of crystallography works well most of the time and accounts for the commanding role of X-ray structure determination as a definitive analytical tool. However, it can fail for a number of reasons.

One is quite simply that the information content of the diffraction amplitudes is so low that there is no clear 'best' match between the observed and computed amplitudes. This situation could arise when the crystals themselves are of a very poor and inconsistent quality. The most naive solution to this situation is to let the structure refine to give a best match and hope that the standard uncertainties on the parameters will warn future users that the analysis was sub-optimal. However, better strategies are available. Weighting schemes have been devised to reduce the impact of observations which probably have more than just random errors and thus make the minimisation procedure more robust. In addition, features in the model can be made to conform to some preconceived ideas. For example, the analyst may have some idea about the values of inter-atomic distances. Inputting this extra information, either as restraints or constraints, can be both powerful and dangerous. The strength is that adding information should tighten up the minimisation function leading to a more acceptable model. The danger is that incorrect assumptions may be imposed on the model.

Another reason for the simple well-located atomistic model to fail is that in the real, actual, crystal, the atoms are not well located. It is now quite evident that the *solid state* is not always a *static state*. The evidence from variable temperature X-ray diffraction experiments and from solid state nmr is that at room temperatures and some times well below, atoms even in molecular materials can undergo large displacements from their mean positions. The traditional models for these displacements are the isotropic and the anisotropic adps. More sophisticated models for the atomic displacements include higher order expansions , but these introduce ever more parameters to be refined against a limited amount of reliable data. Since the diffraction experiment works with samples containing many millions of unit cells, there is always the possibility of spatial inhomogeneity, leading to a diffraction effect which is also space-averaged.

There are a number of signs that the ordinary anisotropic adps may be unsatisfactory approximations to the atomic displacements in the real crystal structure. The most evident is one or two very long axes to the ellipsoid, while the third axis has a quite normal value. The traditional response to this situation is to replace the single atom with its single elongated adp by two atoms lying towards the ends of this axis, each with more isotropic adps. If the two atomic sites become more or less discernable in an electron density map phased by this model, there is a reasonable chance that normal least squares refinement will proceed stably. This situation is often seen in crown ethers, where there are two clear alternative potential locations for the ziz-zag ring structure. If the electron density remains an elongated crest, then refinement will almost certainly only proceed to a reasonable physical model if constraints or restraints are applied to the model. Typically, these would be to restrain inter atomic distances (either to identical but variable values or to prescribed values), and to constrain or restrain the adps of the 'split' atom to be identical or similar.

Another sign that the electron density in the crystal cannot be simply modelled by the normal atomic model is the persistent recurrence of electron density residues in the difference density map. This can of course be due to errors in the observed data (e.g. absorption, anisotropic scaling errors, anisotropic variations in crystal quality, anisotropic integration errors). However, if the data has been collected on a modern instrument with a sufficiently high redundancy in the observations[2], it may be an indication that the well-located atomic model is inappropriate. This situation can occur when there is a guest molecule located in a lattice whose form is largely determined by a host structure. Extended-lattice materials (eg zeolites) commonly show this phenomenon, but it can also be found in crystals of molecular materials where there is a small counter ion or solvent of crystallisation.

The most commonly used method for dealing with this situation is to used a multiply disordered cluster of partially occupied atom sites. The principal attractions of this method are that it can be applied by all commonly available structure analysis programs, and it yields an atomic model – which referees and users may find comfortingly familiar.

In this approach, one is adding ever-more terms into a complex expression in the hope that eventually they will model the average electron distribution as seen by X-rays[3]. If the atoms being added in can be related to each other in a way which makes physical sense, this approach has some well-based justification. However, it can lead to a complex model which may have no real value. It *is* important to be able to model the whole structure reliably[4], since a modelling error in one place will lead to a systematic error in Fc, and hence lead to shifts in all other parameters as they are adjusted to minimise the error in $(Fo\text{-}Fc)^2$ or $(Fo^2\text{-}Fc^2)^2$. However, the atomic model may not be the best model for this kind of problem.

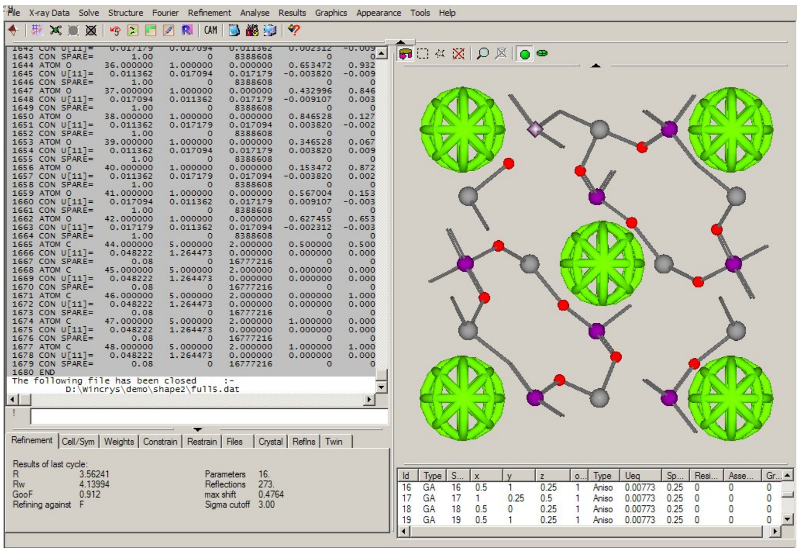
Two other models are easily available to users of CRYSTALS. Neither is unique to CRYSTALS, but neither is commonly and easily available to the ordinary crystallographer in other programs.

One model is to say that the ordinary point atoms spread out by a Gaussian smearing function is just a special case of more general models. In 1950 King & Lipscomb proposed that the electron density could, in suitable circumstances, be regarded as lying on a hollow shell, and thus could be modelled by a suitable Bessel function. This and related ideas have been re-postulated from time to time ever since, but only implemented in programs for local distribution. Funded by a European Union Human Mobility grant, Ludger Schroeder has re-implemented the strategies outlined by Chernyshev, Zhukov, Yatsenko, Aslanov & Shenk into CRYSTALS in such a way that electron density distributed uniformly along a line, around an annulus or over a hollow shell can be freely refined alongside conventional atoms. The spherical shell, for example, can be used to model a freely rotating C60 fragment, or with an additional atom at its centre, to model a tumbling PF_6 counter ion. The line, annulus or shell are infinitely thin (by analogy with point atoms), and have an interaction with the incident X-rays given by the conventional atomic form factor, site occupation factor and isotropic adp. The refineable parameters are the centroid of the figure, the length of the line or radius of the annulus and sphere, and the direction of the line and normal to the annulus, together with site occupancies and isotropic adps. Because these special figures can be mixed freely with conventional atoms, the user has considerable flexibility in building up models for highly disordered fragments. For example, the model could consist of a central atom surrounded by concentric shells of different radii. The hindered rotor described by Bennett, Hutchenson & Foxman has not yet been implemented, but can be approximated by a hollow shell containing embedded partial atoms. The advantage of these models over heavily disordered atomic models is that very few extra parameters need be introduced to achieve an adequate modelling of what is in reality an un-analysable smear of electron density.

[2] In my opinion, high redundancy is much more important than high completeness in 'routine' structure analyses. High redundancy enables the cross-scaling software (Scalepack, Sortav, Sadabs) to make a much better job of correcting for spatially oriented defects in the data. A wedge or shell of missing data, especially at high angles, has little impact on the analysis.

[3] Just as a square wave can be approximated by a sufficiently high-order Fourier series

[4] It is for this same reason that structure analysts labour so long over hydrogen atoms. If they cannot be refined from the observed data, then their significance is marginal. However, omitting them altogether from an analysis leads to a bias in Fc, and hence in the other refined parameters. Just how much labour their positioning justifies in terms of the effect on other parameters does not seem to have been systematically surveyed.



```
1642 CON U[11]=   0.017179   0.017094   0.011362   0.002312  -0.009
1643 CON SPARE=       1.00          0    8388608          0      0
1644 ATOM O      36.000000   1.000000   0.000000   0.653472   0.932
1645 CON U[11]=   0.011362   0.017094   0.017179  -0.003820  -0.009
1646 CON SPARE=       1.00          0    8388608          0      0
1647 ATOM O      37.000000   1.000000   0.000000   0.432996   0.846
1648 CON U[11]=   0.017094   0.011362   0.017179  -0.009107   0.003
1649 CON SPARE=       1.00          0    8388608          0      0
1650 ATOM O      38.000000   1.000000   0.000000   0.846528   0.127
1651 CON U[11]=   0.011362   0.017179   0.017094   0.003820  -0.002
1652 CON SPARE=       1.00          0    8388608          0      0
1653 ATOM O      39.000000   1.000000   0.000000   0.346528   0.067
1654 CON U[11]=   0.011362   0.017094   0.017179   0.003820   0.009
1655 CON SPARE=       1.00          0    8388608          0      0
1656 ATOM O      40.000000   1.000000   0.000000   0.153472   0.872
1657 CON U[11]=   0.011362   0.017179   0.017094  -0.003820   0.002
1658 CON SPARE=       1.00          0    8388608          0      0
1659 ATOM O      41.000000   1.000000   0.000000   0.567004   0.153
1660 CON U[11]=   0.017094   0.011362   0.017179   0.009107  -0.003
1661 CON SPARE=       1.00          0    8388608          0      0
1662 ATOM O      42.000000   1.000000   0.000000   0.627455   0.653
1663 CON U[11]=   0.017179   0.011362   0.017094  -0.002312  -0.003
1664 CON SPARE=       1.00          0    8388608          0      0
1665 ATOM C      44.000000   5.000000   2.000000   0.500000   0.500
1666 CON U[11]=   0.048222   1.264473   0.000000   0.000000   0.000
1667 CON SPARE=       0.08          0   16777216          0      0
1668 ATOM C      45.000000   5.000000   2.000000   0.000000   0.000
1669 CON U[11]=   0.048222   1.264473   0.000000   0.000000   0.000
1670 CON SPARE=       0.08          0   16777216          0      0
1671 ATOM C      46.000000   5.000000   2.000000   0.000000   1.000
1672 CON U[11]=   0.048222   1.264473   0.000000   0.000000   0.000
1673 CON SPARE=       0.08          0   16777216          0      0
1674 ATOM C      47.000000   5.000000   2.000000   1.000000   0.000
1675 CON U[11]=   0.048222   1.264473   0.000000   0.000000   0.000
1676 CON SPARE=       0.08          0   16777216          0      0
1677 ATOM C      48.000000   5.000000   2.000000   1.000000   1.000
1678 CON U[11]=   0.048222   1.264473   0.000000   0.000000   0.000
1679 CON SPARE=       0.08          0   16777216          0      0
1680 END
The following file has been closed     :-
          D:\Wincrys\demo\shape2\full5.dat
```

```
Results of last cycle:
R                 3.56241          Parameters      16.
Rw                4.13994          Reflections     273.
GooF              0.912            max shift       0.4764
Refining against  F                Sigma cutoff    3.00
```

Id	Type	S...	x	y	z	o...	Type	Ueq	Sp...	Resi...	Asse...	Gr...
16	GA	16	0.5	1	0.25	1	Aniso	0.00773	0.25	0	0	0
17	GA	17	1	0.25	0.5	1	Aniso	0.00773	0.25	0	0	0
18	GA	18	0.5	0	0.25	1	Aniso	0.00773	0.25	0	0	0
19	GA	19	0.5	1	0.25	1	Aniso	0.00773	0.25	0	0	0

Fig 1: *Screenshot of CRYSTALS showing spherically disordered guest molecule. ZnGaPO is a templated phosphate framework structure produced hydrothermally in space group P -4 3 n (218). The template, piperazine, lies at Wyckoff position a (23), so is necessarily disordered. The radius of the sphere (1.26A) corresponds closely to a C1-C3 distance (1.25A). A. Chippendale & A. Cowley, in preparation.*

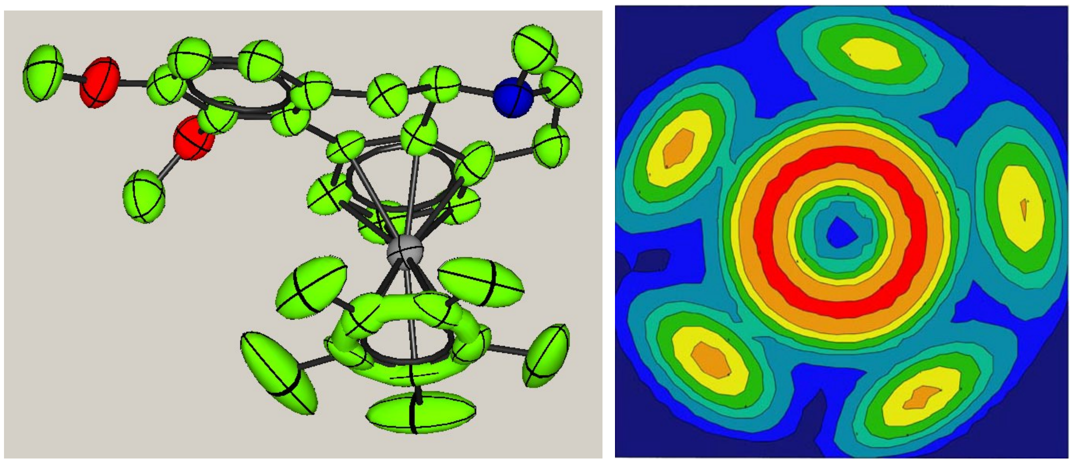


Fig 2: *a) CRYSTALS screen shot of Cp* complex, with inner ring of atoms modelled by elliptical adps embedded in an annulus, and b) corresponding electron density*

However, if the smear of electron density computed from the best estimates of phases really is un-interpretable, perhaps the most honest approach to the analysis is to include it in the model as exactly that. This is the tactic adopted by Ton Speck in his superb SQUEEZE program.

The structure factor is a complex number (has both magnitude and phase). The magnitude can be represented by:

$$F^2 = A^2 + B^2$$

where A is the real and B the imaginary part. From the continuous electron density, A can be computed from:

$$A_{hkl} = \int_v \rho_{xyz} \cos 2\pi(hx + ky + lz)\partial v$$

For a discrete atom model, A can be computed from:

$$A_{hkl} = \sum_j f_j \cos 2\pi(hx + ky + lz)$$

with B given by similar sin terms.

Ton has tried to promote a hybrid structure factor expression:

$$A_{hkl} = \sum_j f_j \cos 2\pi(hx + ky + lz) + \int_v \rho_{xyz} \cos 2\pi(hx + ky + lz) \partial v$$

The first term is a summation over the resolved atoms. The integral in the second term is replaced by a summation over unresolved parts of the electron density map, with a similar expression for B.

In this method, a Fourier map is computed based on whatever phases are available. The unit cell is searched for voids large enough to potentially contain atoms or molecules. The calculated electron density in these void volumes can then be reverse-transformed into structure factor contributions[5]. In the normal application of this method, the structure factor amplitudes computed from the density in the void are subtracted from the observed structure factor amplitudes, and the residue used as the target for further refinement. This is evidently far from ideal, since the phases of the contributions from the electron density map are not used. In CRYSTALS, the transform of the void density is save for each reflection as the A and B parts, to be used in the computation of Fc and the phase angle together with the A and B parts from the discrete atoms. This means that the strategy does not tinker with the observations, has no lasting effect on the model (the A and B parts from the void can be dismissed as required), but more importantly, since they contribute to the phase angles, they lead to improved electron density maps.

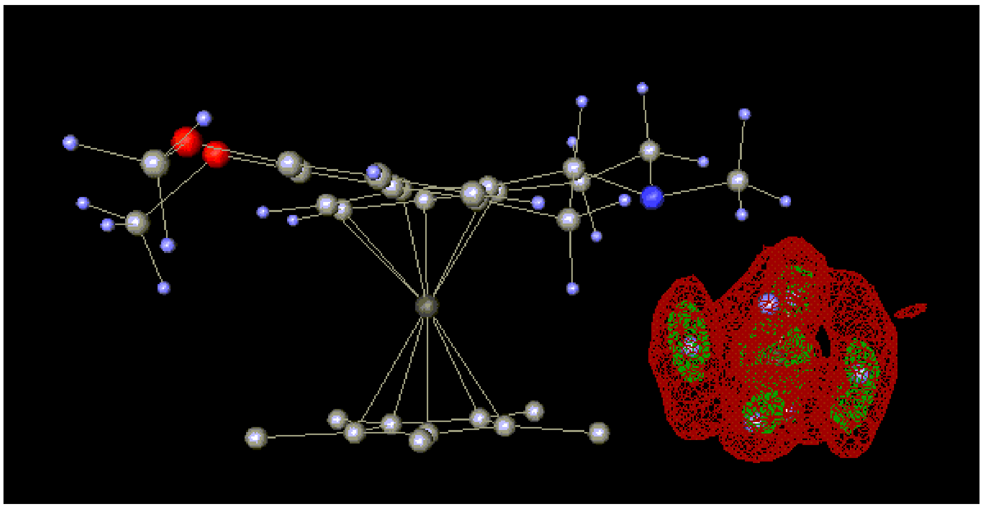


Fig 3: *The PF_6 counter ion in the above complex represented by its computed electron density. The atoms embeded within the density were given occupation factors of zero so that they do not contribute to the structure factors, and are just there to illustrate that the electron density does envelop the counter ion. (Illustration by Michal Husak's MCE viewer – available via CCP14 website http://www.ccp14.ac.uk/)*

These strategies have been seamlessly integrated in CRYSTALS so that users can mix them in whatever way they choose. My hope is to see other programs adopt similar ideas, and to see an end-user community less uneasy about accepting non-atomic representations of crystal structures. The benefit will be that less time is spent trying to fit almost valueless atomic models to volumes of extreme disorder without degrading the quality of the rest of the analysis.

[5] In macromolecular crystallography, the whole of the structure factor and derivative calculation is based on back-transforming a continuous electron density distribution in the cell built up from atomic contributions.

Writing Binary Data

Scott A. Belmonte,
91 Lord Nelson Street, Warrington, Cheshire, WA1 2JF, U.K. E-mail: scott.belmonte@ntlworld.com - WWW: http://www.ccp14.ac.uk/ccp/web-mirrors/scott-belmonte-software/

Abstract

In this article, techniques for writing binary data files are detailed. Common issues when writing binary data are highlighted and examples in FORTRAN and C are given.

Introduction

This article is the sequel to the “Reading Binary Data” article in the January 2004 edition of this news letter [1]. Much of the introductory material in that article is applicable to writing binary data and should be read in conjunction with this article. Specifically, the information on endian and file formats apply to both reading and writing binary data.

As with reading binary data, the biggest problem a programmer faces when writing binary data is knowing the format of the data to write. Ideally, documentation will be available giving details of the file format of interest. For example, widely used image formats such as bmp, gif and jpg are well documented. However, proprietry formats may deliberately be kept secret. In this case, analysis of an example file using a hex dump utility and a bit of trial and error may be fruitful. Of course, if the programmer just wants to devise a file format to be used by programs under his control then handling binary data is easy, as will be shown in the rest of this article.

Writing Binary Data (FORTRAN)

Writing binary data basically comprises converting internal data types, like INTEGER and REAL, into their byte representations and storing the bytes in a buffer in memory. Once the buffer has been constructed, it can be written to a file.

The following routine shows how to convert a REAL to its byte representation and save it to a byte BUFFER (the “writebin” software available at the web site above contains conversion routines for other types):

```
C
C**********************************************************************
C     Routine: WRITE_BIN_REAL
C
C     Description:
C        Writes a 32-bit real to BUFFER. The bytes will be swapped
C        if SWAP is true. The function returns the number of bytes
C        written to BUFFER.
C
C**********************************************************************
      INTEGER FUNCTION WRITE_BIN_REAL(BUFFER, DATA, SWAP)
      IMPLICIT NONE
C
C     Parameters
C
      BYTE    BUFFER(4)
      REAL    DATA
      LOGICAL SWAP
C
C     Variables
C
      BYTE          TMPBUF(4)
      REAL          R4
      EQUIVALENCE (TMPBUF, R4)
```

```
C
      R4 = DATA
C
      IF (SWAP) THEN
         BUFFER(1) = TMPBUF(4)
         BUFFER(2) = TMPBUF(3)
         BUFFER(3) = TMPBUF(2)
         BUFFER(4) = TMPBUF(1)
      ELSE
         BUFFER(1) = TMPBUF(1)
         BUFFER(2) = TMPBUF(2)
         BUFFER(3) = TMPBUF(3)
         BUFFER(4) = TMPBUF(4)
      ENDIF
C
      WRITE_BIN_REAL = 4
      RETURN
      END
C
```

BUFFER points to the location to write the DATA to. The data are converted into bytes using an EQUIVALENCE and the bytes are copied to the buffer. If the SWAP variable is .TRUE. then the byte order is swapped before copying to the buffer. If the endian of the file is different to the native endian of the CPU then the byte order needs to be swapped for the file to be written properly. (See [1] for a discussion on "Endian".)

The routine returns the number of bytes written to the buffer. This can be used by the caller to update the position of then next write into the buffer, as will be shown in the next example.

It should be noted that the conversion routines are inherently non-portable. For example, REAL is not necessarily a 4-byte quantity on all machines. If problems are encountered then the compiler documentation should be checked to find out the sizes of each type and the conversion routines updated accordingly.

The following code is an example writing a ficticious image file with the format:

Bytes 1, 2: Unsigned 16-bit integer containing image width.
Bytes 3, 4: Unsigned 16-bit integer containing image height.
Bytes 5 to (width*height*size of real + 4): Image data as REALs.

```
      PROGRAM WRITE_BIN_EX1
      IMPLICIT NONE
C
C     Functions
C
      INTEGER WRITE_BIN_UINT16
      INTEGER WRITE_BIN_REAL
      INTEGER WRITE_BIN_FILE
C
C     Variables
C
      INTEGER   REAL_SIZE           ! Size of REAL in bytes
      PARAMETER(REAL_SIZE = 4)
C
      INTEGER   UINT16_SIZE         ! Size of UINT16 in bytes
      PARAMETER(UINT16_SIZE = 2)
C
      INTEGER   WIDTH
      PARAMETER(WIDTH = 480)
C
      INTEGER   HEIGHT
      PARAMETER(HEIGHT = 640)
C
      INTEGER   HDR_SIZE            ! Header size (2 UINT16s)
      PARAMETER(HDR_SIZE = 2*UINT16_SIZE)
C
```

```
      INTEGER   FILELEN             ! Length of file in bytes
      PARAMETER(FILELEN = HDR_SIZE + WIDTH*HEIGHT*REAL_SIZE)
C
      BYTE      BUFFER(FILELEN)     ! Buffer to hold the file
      REAL      DATA(WIDTH*HEIGHT) ! Data to be written
      INTEGER   I, INDEX
C
C     Initialise buffer
C
      DO I = 1, FILELEN
          BUFFER(I) = 0
      ENDDO
C
C     Fill DATA with an arbitrary number, 3.1416,
C     for this example
C
      DO I = 1, WIDTH*HEIGHT
          DATA(I) = 3.1416
      ENDDO
C
C     Write the width and height to the buffer as
C     unsigned 16-bit (2 byte) integers. Then write
C     the data to the buffer. The byte order is not
C     swapped.
C
      INDEX = 1
      INDEX = INDEX + WRITE_BIN_UINT16(BUFFER(INDEX), WIDTH, .FALSE.)
      INDEX = INDEX + WRITE_BIN_UINT16(BUFFER(INDEX), HEIGHT, .FALSE.)
      DO I = 1, WIDTH*HEIGHT
        INDEX = INDEX + WRITE_BIN_REAL(BUFFER(INDEX), DATA(I), .FALSE.)
      ENDDO
C
C     Write buffer to a file.
C
      IF (WRITE_BIN_FILE('example.dat', BUFFER, FILELEN) .NE. 0)
     $    GOTO 901
      RETURN
C
C     Error traps
C
 901  WRITE(*,*) '** Error writing binary file!'
      RETURN
      END
C
```

WRITE_BIN_UINT16 is similar to WRITE_BIN_REAL (see the writebin code at the web site above for the implementation).

The INDEX variable holds the position of the next byte in the buffer to write. The return value of the WRITE_BIN_UINT16 and WRITE_BIN_REAL functions is used to update the INDEX variable by the appropriate amount according to the size of the data just written.

The WRITE_BIN_FILE routine writes the buffer to a file. It takes the name of the file to write, the buffer and the number of bytes to write.

```
C*********************************************************************
C     Routine: WRITE_BIN_FILE
C
C     Description:
C        Writes NUMBYTES bytes from BUFFER to a binary file called
C        NAME. Returns non-zero if the write fails.
C
C*********************************************************************
      INTEGER FUNCTION WRITE_BIN_FILE(NAME, BUFFER, NUMBYTES)
      IMPLICIT NONE
C
C     Parameters
C
      CHARACTER*(*) NAME        ! File name
      INTEGER        NUMBYTES     ! Number of bytes in buffer
      BYTE           BUFFER(NUMBYTES)    ! The data to write
C
C
      OPEN(UNIT=11, FILE=NAME, STATUS='UNKNOWN', FORM='UNFORMATTED',
     $     ACCESS='DIRECT', RECL=NUMBYTES, ERR=901)
C
      WRITE(UNIT=11, REC=1, ERR=902) BUFFER
      CLOSE(UNIT=11)
      WRITE_BIN_FILE = 0
      RETURN
C
C     Error trap
C
 901  WRITE(*,*) '** Error opening file for output: ', NAME
      WRITE_BIN_FILE = 1
      RETURN
 902  WRITE(*,*) '** Error while writing file: ', NAME
      WRITE_BIN_FILE = 2
      RETURN
      END
C
```

The routine writes the buffer to a file in a single unformatted record. The record length (RECL) is set to the buffer length in bytes. N.B. RECL on some compilers can be the length of a record in words. On 32-bit machines, a word is normally defined as 4 bytes. When writing a file, if its length is larger than expected then try changing the OPEN statement above to:

```
      OPEN(UNIT=11, FILE=NAME, STATUS='UNKNOWN', FORM='UNFORMATTED',
     $     ACCESS='DIRECT', RECL=NUMBYTES/4, ERR=901)
```

Writing Binary Data (C)

The C standard library provides functions that can be used to write binary files: fopen, fwrite and fclose.

fopen takes the file name and the open mode and returns a handle to the opened file, or NULL if the file open failed. An open mode of “wb” means open a binary file for writing.

```
#include <stdio.h>

    FILE *file_p;
    file_p = fopen("example.dat", "wb");
```

fwrite takes a pointer to the data to write, the size (in bytes) of an individual item of data, the number of items of data to write and the handle to the file to write to. fwrite returns the number of items written. This might be less than the number of items requested to be written if an error occurred during the write.

```
    num_items_written = fwrite(data_p, item_size, item_count, file_p);
```

fclose takes a file handle and closes the file.

```
    fclose(file_p);
```

fwrite can be used to write any type of data. It can be used to write C structures directly but this is not recommended since the way structure members are arranged in memory is compiler dependent. Each member should be written individually to ensure that the file is written in the manner the programmer intended.

Some file formats require that the data have a particular endian. If the endian of the file is different to the native endian of the CPU then the byte order of the data needs to be swapped before the data are written to disc. (See [1] for a discussion on “Endian”.)

The following function wraps the fwrite function and swaps the data if required.

```
/*************************
 * writebin.h
 */
#ifndef WRITEBIN_H
#define WRITEBIN_H

#include <stdio.h>

typedef enum
{
    DONT_SWAP,
    SWAP
} e_swap;

size_t write_bin_file(const void *data_p,
                      size_t      item_size,
                      size_t      item_count,
                      FILE       *stream_p,
                      e_swap      swap);

#endif /* WRITEBIN_H */


/*************************
 * writebin.c
 */
#include <stdlib.h>
#include "writebin.h"

/****************************************************************************
 *
 * Function: write_bin_file
 *
 * Description: This function wraps the standard library function fwrite().
 *              It swaps the bytes of the raw data if the parameter swap
 *              is SWAP and item_size is either 2, 4 or 8. If item_size
 *              is not 2, 4 or 8 then the raw data are written without
 *              swapping.
 *
 * Input: data_p     - Pointer the buffer containing the data to be written.
 *        item_size  - The size in bytes of an individual item to be written.
 *        item_count - The number of items to write.
 *        stream_p   - File handle of an open file.
 *        swap       - If SWAP then swap the bytes in the buffer,
 *                       If DONT_SWAP then don't swap bytes.
 *
 * Output: size_t - The actual number of items (not bytes) written. May be
 *                  less than the number requested if an error occurred
 *                  while writing the file.
 *
 ****************************************************************************/
size_t write_bin_file(const void *data_p,
                      size_t      item_size,
                      size_t      item_count,
                      FILE       *stream_p,
                      e_swap      swap)
{
    const unsigned char *orig_data_p;
    unsigned char       *swapped_data_p;
    unsigned char       *pointer;
    size_t               num_items_written;
    size_t               i;

    num_items_written = 0;

    if (stream_p != NULL && data_p != NULL)
    {
        orig_data_p = (const unsigned char *) data_p;

        /* Swap bytes if asked and if item_size is divisible by two. */
        if (swap == SWAP && (item_size % 2 == 0))
        {
            swapped_data_p = (unsigned char *) malloc(item_size*item_count);
            pointer        = swapped_data_p;

            if (swapped_data_p != NULL)
            {
                switch (item_size)
                {
                case 2:
                    for (i = 0; i < item_count; i++)
                    {
                        *(pointer)     = *(orig_data_p + 1);
                        *(pointer + 1) = *(orig_data_p);
                        pointer     += item_size;
                        orig_data_p += item_size;
                    }
                    break;

                case 4:
                    for (i = 0; i < item_count; i++)
                    {
                        *(pointer)     = *(orig_data_p + 3);
                        *(pointer + 1) = *(orig_data_p + 2);
                        *(pointer + 2) = *(orig_data_p + 1);
                        *(pointer + 3) = *(orig_data_p);
                        pointer     += item_size;
                        orig_data_p += item_size;
                    }
                    break;

                case 8:
                    for (i = 0; i < item_count; i++)
                    {
                        *(pointer)     = *(orig_data_p + 7);
                        *(pointer + 1) = *(orig_data_p + 6);
                        *(pointer + 2) = *(orig_data_p + 5);
                        *(pointer + 3) = *(orig_data_p + 4);
                        *(pointer + 4) = *(orig_data_p + 3);
                        *(pointer + 5) = *(orig_data_p + 2);
                        *(pointer + 6) = *(orig_data_p + 1);
                        *(pointer + 7) = *(orig_data_p);
                        pointer     += item_size;
                        orig_data_p += item_size;
                    }
                    break;

                default:
                    /* Issue warning message and then write data without swapping */
                    fprintf(stderr, "write_bin_file: Cannot swap data. ");
                    fprintf(stderr, "Unsupported data size: %d\n", item_size);
                    for (i = 0; i < item_count*item_size; i++)
                    {
                        *pointer++ = *orig_data_p++;
                    }
                    break;
                }

                /* Write the swapped items */
                num_items_written =
                    fwrite(swapped_data_p, item_size, item_count, stream_p);
                free(swapped_data_p);
            }
        }
        else
        {
            /* Write the data directly */
            num_items_written = fwrite(data_p, item_size, item_count, stream_p);
        }
    }

    return num_items_written;
}
```

The following code is an example writing a ficticious image file with the format:

Bytes 0, 1: Unsigned 16-bit integer containing image width.
Bytes 2, 3: Unsigned 16-bit integer containing image height.
Bytes 4 to (width*height*size of float + 3): Image data as float.

```
/*************************
 * writebinex1.c
 */
#include <stdlib.h>
#include "writebin.h"

typedef float data_type;

int main()
{
    FILE           *file_p;
    unsigned short  width;
    unsigned short  height;
    data_type      *data_p;
    int    i;

    width  = 480;
    height = 640;
    data_p = (data_type *) malloc(width*height*sizeof(data_type));

    if (data_p != NULL && (file_p = fopen("example.dat", "wb")) != NULL)
    {
        /* Initialise data with an arbitrary number, 3.1416 for this example */
        for (i = 0; i < width*height; i++)
        {
            data_p[i] = (data_type) 3.1416;
        }

        if (write_bin_file(&width, sizeof(width), 1, file_p, DONT_SWAP) != 1)
        {
            fprintf(stderr, "Error writing width!\n");
            fclose(file_p);
            exit(1);
        }

        if (write_bin_file(&height, sizeof(height), 1, file_p, DONT_SWAP) != 1)
        {
            fprintf(stderr, "Error writing height!\n");
            fclose(file_p);
            exit(1);
        }

        if (write_bin_file(data_p,
                           sizeof(data_type),
                           width*height,
                           file_p, DONT_SWAP) != width*height)
        {
            fprintf(stderr, "Error writing data!\n");
            fclose(file_p);
            exit(1);
        }

        fclose(file_p);
    }
    else
    {
        fprintf(stderr, "Error writing file!\n");
        exit(1);
    }

    return 0;
}
```

Conclusion

Methods for writing binary data files in FORTRAN and C have been outlined. Using the tools in this article it should be possible to write any binary file format.

The source code in this article can be found under the ‘writebin’ directory at:
http://www.ccp14.ac.uk/ccp/web-mirrors/scott-belmonte-software/

References

[1] “Reading Binary Data”, IUCr Compcomm January 2004, http://www.iucr.org/iucr-top/comm/ccom/newsletters/2004jan/ .

Scientific Programming. The .NET case

Nikos Kourkoumelis,
Department of Physics, University of Ioannina, 45110 Ioannina, Greece - Email : nkourkou@cc.uoi.gr ; WWW: http://users.uoi.gr/nkourkou/

Introduction

The Microsoft .NET Framework [1] is a platform for building, deploying, and running applications and web services. It provides a standards-based, multi-language environment where language interoperability and machine independent programming intersect. Scientific community has therefore an alternative to integrate existing investments of source code with modern application design.

.NET Architecture and Execution Model

The Microsoft .NET framework is a new development platform with a new programming interface that provides APIs better suited to modern programmer needs, featuring inter-language and inter-machine interoperability. The .NET Framework has two main components: the .NET Framework Class Library (FCL) and the Common Language Runtime (CLR). The .NET framework base classes are a collection of built-in functions, objects, properties and methods that can be utilized by any .NET compatible language. The common language runtime is the foundation of the .NET framework and is responsible for the execution and management of .NET applications, as well as the compilation of .NET applications into native code. The CLR is the environment under which .NET applications are run. From this perspective, the CLR is the part of .NET that supports managed code; that is well-behaved code in terms of multi language support, deployment, security, portability, scalability and functionality.

All common language runtime–compliant source code languages compile to the same intermediate language (IL) and not immediately to native code. A second step is required, called just-in-time (JIT) compilation. A tool called just-in-time compiler, or JITter, reads the IL and translates it into instructions for the machine on which it is executing. This translation happens on a method-by-method basis, thus avoiding translating large quantities of code that might never be required. Although the use of IL imposes a small start-up overhead, it provides .NET with a certain amount of platform independence, as long as each platform can have its suitable JITter. Currently, the vast majority of applications supporting .NET are for Windows OS while a free implementation of the .NET Development Framework for Linux is under development [2]. Moreover, a shared, open source implementation of CLR API, called the Common Language Infrastructure, is provided by Microsoft and has been adapted to Windows, FreeBSD and MacOS X operating systems, under the name of Rotor (or SSCLI) [3].

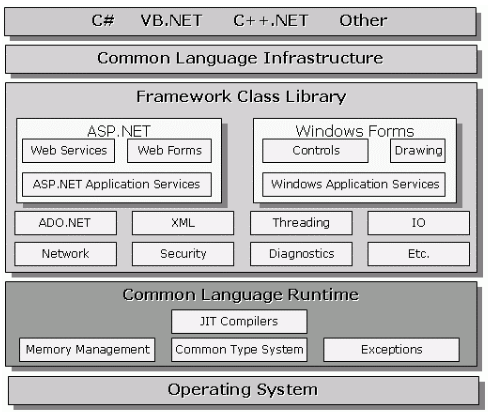


Fig. 1: *The .NET Framework sits on top of the OS*

Schematically the .NET framework is presented in Fig.1, where it is clear that besides being responsible for application loading and execution, the CLR provides additional services such as: object lifetime and memory management, exception handling (even across languages), interoperation between managed and unmanaged code and type safety.

Component Model – Example

.NET programming languages are fully object-oriented, which means they support the four basic tenets of object-oriented programming (OOP): abstraction, encapsulation, inheritance and polymorphism. OOP is playing a major role in crystallography mostly through reusable modules programmed in Fortran and C++ [4]. In .NET terminology these are called assemblies. As an example, PowDLL [5] is a useful .NET assembly for the interconversion procedure between variable formats of powder diffraction files. It is utilized as a reusable dynamic link library imported in any .NET language using the “PowDLL.PowderFileTypes” statement which is the fully qualified type name of the assembly. Upon declaration of an appropriate object, three boolean public methods are exposed which return *True* when no exception is raised. These methods which are: DoFileConversion(Input, Output, ShowError), LoadDataFromFile(Input, FileType, ShowError) and WriteDataToFile(Output, FileType, ShowError) can be accessed by any external module that references the library. If an error occurs, the System.Exception object, which is the centralized error handler, takes control and shows a proper message keeping the process alive. The following table shows part of the relevant source code written in VB.NET.

```
#Region "Dll Interface functions" 'Not all code functionality is shown
Public Function DoFileConversion(ByVal Source As String, ByVal_ Destination As String, Optional
ByVal sE As ShowErrors = ShowErrors.DontShowErr) As Boolean
        Dim tmpOb As Object
        'first tmpOB is passed by reference
        If LoadDataFromFile(Source, tmpOb, sE) = True Then
            ' here the loaded object will be written to a file.
            If WriteDataToFile(tmpOb, Destination, sE) = True Then
                Return True
            End If
        End If
        Return False
    End Function
Public Function LoadDataFromFile(ByVal sFile As String, ByRef q As Object, ByVal sE As
ShowErrors) As Boolean
        Dim extS As String = ExtractFileExtention(sFile)
        Select Case extS
            Case "xy"
                Dim tmpOb As fileXY
                q = tmpob
            Case Else
                If sE = ShowErrors.ShowErr Then
                    MsgBox("*." + extS + " filetype is not supported")
                End If
                Return False
        End Select
        If q.FromFileToObject(sFile, sE) = 2 Then Return True
        Return False
    End Function
Public Function WriteDataToFile(ByVal Sobj As Object, ByVal dFile As String, ByVal sE As
ShowErrors) As Boolean
        Dim ObjectToWrite As Object
        Dim extD As String = ExtractFileExtention(dFile)
        Select Case extD
            Case "xy"
                Dim tmpObj As fileXY
                ObjectToWrite = tmpObj
            Case Else
                If sE = ShowErrors.ShowErr Then
                    MsgBox("*." + extD + " filetype is not supported")
                End If
                Return False
        End Select
        If ObjectToWrite.FromObjectToFile(dFile, sE) = 2 Then Return True
        Return False
    End Function
#End Region
```

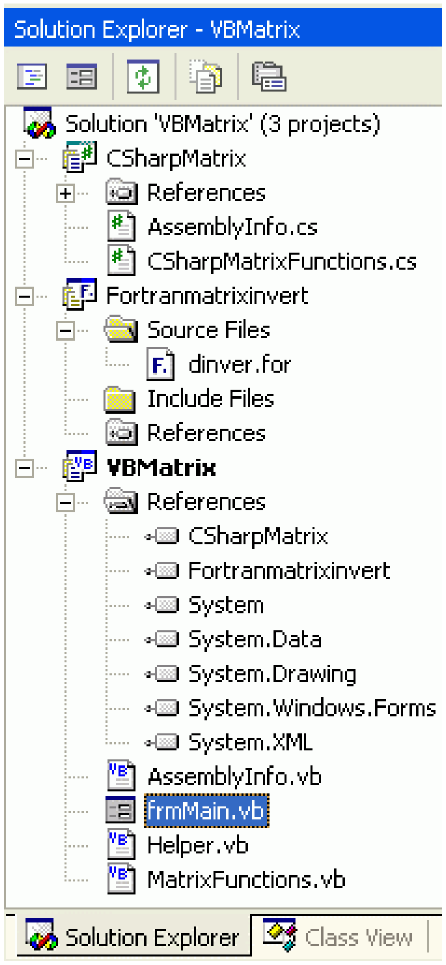
Every powder file type acts as a private structure (encapsulation):

```
#Region "*.xy" 'Plain XY File Format
Private Structure fileXY
        Dim LStart, LStop, LStep, y(), Alpha1, Alpha2, Ratio As Decimal
Public Function FromFileToObject(ByVal Fname As String, Optional ByVal sh As ShowErrors =
ShowErrors.DontShowErr) As Byte
        Try
             Dim sr As StreamReader = New StreamReader(Fname)
             Dim tmpLine As String
                 tmpLine = sr.ReadToEnd
                'now splits it into
             Dim tmpPIN() As Decimal
                If Not CutStringIntoIntegers(tmpLine, tmpPIN) Then
                  sr.Close()
                  Return 1
                End If
                LStep = tmpPIN(2) - (tmpPIN(0))
                LStart = tmpPIN(0)
                LStop = tmpPIN(tmpPIN.GetUpperBound(0) - 1)
                Alpha1 = 0
                Alpha2 = 0
                Ratio = 0
                ReDim y((tmpPIN.GetUpperBound(0) \ 2))
                Dim i, j
                j = 0
                For i = 1 To tmpPIN.GetUpperBound(0) Step 2
                    y(j) = tmpPIN(i)
                    j += 1
                Next
                sr.Close()
             Catch ex As Exception
                If sh = ShowErrors.ShowErr Then
                    MsgBox(ex.Message)
                End If
                Return 1
        End Try
       Return 2
 End Function
Public Function FromObjectToFile(ByVal Fname As String, Optional ByVal sh As ShowErrors =
ShowErrors.DontShowErr) As Byte
        Try
             Dim sw As StreamWriter = New StreamWriter(Fname)
             Dim tmpXvalue As Decimal
                 tmpXvalue = LStart
             Dim i
                For i = 0 To y.GetUpperBound(0)
                 sw.WriteLine(tmpXvalue.ToString & " " & y(i).ToString)
                 tmpXvalue += LStep
                Next i
                sw.Close()
             Catch ex As Exception
                If sh = ShowErrors.ShowErr Then
                    MsgBox(ex.Message)
                End If
       Return 1
        End Try
       Return 2
 End Function
End Structure
#End Region
```

Language Interoperability – Example

Reusable .NET assemblies exploit an additional unique feature: they permit full language integration providing the possibility to: (i) inherit from classes, (ii) handle thrown exceptions, (iii) debug, (iv) declare variables based on types declared in another language and (v) take advantage of polymorphism across different languages. This is possible because of the shared .NET type system which retains high-level data-type information such as classes and inheritance hierarchies. Once a program is compiled into the .NET architecture, its language of origin disappears and it becomes language neutral. As a consequence, a considerable trend appears towards .NET with a variety of different languages which share their representation and runtime behavior (Fortran, Ada, Perl, Python, Delphi, Smalltalk, Cobol etc.) [6]. In addition, the System.Reflection.Emit namespace provides all the functionality needed for developing new compilers that target the CLR.

This seamless interoperability is one of the most realistic reasons for developers to use, for example, components written in Fortran and program the main application using C#. One of the issues often arise in crystallographic computing is array manipulation which is usually achieved by using optimized, open source, Fortran and C routines. These modules can be used smoothly as part of a .NET project. Fig. 2 illustrates the main GUI written in Visual Basic.NET while the matrix operations modules are a combination of Fortran, VB and C#.



```
Solution Explorer - VBMatrix
Solution 'VBMatrix' (3 projects)
  CSharpMatrix
    References
    AssemblyInfo.cs
    CSharpMatrixFunctions.cs
  Fortranmatrixinvert
    Source Files
      dinver.for
    Include Files
    References
  VBMatrix
    References
      CSharpMatrix
      Fortranmatrixinvert
      System
      System.Data
      System.Drawing
      System.Windows.Forms
      System.XML
    AssemblyInfo.vb
    frmMain.vb
    Helper.vb
    MatrixFunctions.vb
Solution Explorer | Class View
```

Fig. 2: *Mixed Language Environment*

FORTRAN Routine for Matrix Inverse

```
      SUBROUTINE DINVER(A,N,NUS,D,L,M)
      REAL*8 A(*),L(*),M(*)
      REAL*8 BIGA,HOLD
      REAL*8 A,D
      IF(NUS.GT.1)GOTO 1
      A(1)=1.0/A(1)
      D=1.0D0
      RETURN
1     D=1.0D0
      NK=-N
      DO 11 K=1,NUS
         NK=NK+N
         L(K)=K
         M(K)=K
         KK=NK+K
         BIGA=A(KK)
         DO 2 J=K,NUS
            IZ=N*(J-1)
            DO 2 I=K,NUS
             IJ=IZ+I
```

```
            IK=NK+I
            A(IK)=A(IK)/(-BIGA)
8           CONTINUE
         DO 9 I=1,NUS
            IK=NK+I
            HOLD=A(IK)
            IJ=I-N
            DO 9 J=1,NUS
               IJ=IJ+N
               IF(I.EQ.K)GOTO 9
               IF(J.EQ.K)GOTO 9
               KJ=IJ-I+K
               A(IJ)=HOLD*A(KJ)+A(IJ)
9              CONTINUE
         KJ=K-N
         DO 10 J=1,NUS
            KJ=KJ+N
            IF(J.EQ.K)GOTO 10
            A(KJ)=A(KJ)/BIGA
10          CONTINUE
```

```
              IF(ABS(BIGA).GE.ABS(A(IJ)))GOTO 2
               BIGA=A(IJ)
               L(K)=I
               M(K)=J
2              CONTINUE
          J=L(K)
          IF(J.LE.K)GOTO 4
          KI=K-N
          DO 3 I=1,NUS
             KI=KI+N
             HOLD=-A(KI)
             JI=KI-K+J
             A(KI)=A(JI)
3            A(JI)=HOLD
4         I=M(K)
          IF(I.LE.K)GOTO 6
          JP=N*(I-1)
          DO 5 J=1,NUS
             JK=NK+J
             JI=JP+J
             HOLD=-A(JK)
             A(JK)=A(JI)
5            A(JI)=HOLD
6         IF(BIGA.NE.0.0)GOTO 7
          D=0.0
          RETURN
7         DO 8 I=1,NUS
             IF(I.EQ.K)GOTO 8
```

```
             D=D*BIGA
             A(KK)=1.0/BIGA
11           CONTINUE
          K=NUS
12        K=K-1
          IF(K.LE.0)RETURN
          I=L(K)
          IF(I.LE.K)GOTO 14
          JQ=N*(K-1)
          JR=N*(I-1)
          DO 13 J=1,NUS
             JK=JQ+J
             HOLD=A(JK)
             JI=JR+J
             A(JK)=-A(JI)
13           A(JI)=HOLD
14        J=M(K)
          IF(J.LE.K)GOTO 12
          KI=K-N
          DO 15 I=1,NUS
             KI=KI+N
             HOLD=A(KI)
             JI=KI-K+J
             A(KI)=-A(JI)
15           A(JI)=HOLD
          GOTO 12
          END
```

VB.NET Routine for Matrix Transpose

```
Public Class MatrixFunctions
Public Shared Function TransposeMatrix(ByVal A(,) As Double) As Double(,)
        Dim i, j As Integer
        ReDim TransposeMatrix(A.GetUpperBound(0), A.GetUpperBound(1))
        For i = 0 To A.GetUpperBound(0)
            For j = 0 To A.GetUpperBound(1)
                TransposeMatrix.SetValue(A(i, j), j, i)
            Next j
        Next i
End Function
End Class
```

C# Routine for Simple Matrix Addition

```
namespace CSharpMatrix
{
  public class CSharpMatrixFunctions
        {
     public double[,] AddMatrix(double[,] A, double[,] B)
               {
        double[,] ds;
        ds = new double[A.GetUpperBound(0) + 1, A.GetUpperBound(1) + 1];
        if (A.Length == B.Length)
               {
               int i2 = A.GetUpperBound(0);
               for (int i1 = 0; i1 <= i2; i1++)
               {
                       int k = A.GetUpperBound(1);
                       for (int j = 0; j <= k; j++)
                       {
                       ds.SetValue((A[i1, j] + B[i1, j]), i1, j);
               }}      }
        return ds;
}}}
```

Network Services – High Performance Computing

.NET also supports parallel and distributed computing. This challenging programming effort has been so far implemented mostly in Java due to the straightforward management of Java Virtual Machine. A virtual machine (VM) is a program that creates an artificial or abstract computer running on top of an existing computer. The VMs hide the normal computer hardware behind a simpler or different computational model. CLR can be thought of as an advanced design and implementation of a distributed virtual machine for handling the wide range of distributed programming paradigms incorporated in .NET [7] providing a remoting architecture for open Internet standards, including the Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP).

High-performance computing access can be also achieved using .NET architecture by means of tools specifically designed for parallel processing and clustering [8]. Although it is widely believed that VMs give poor computation performance, the above examples as well as the broad class of optimizations performed by JITter [9], suggest the opposite.

In general, an undoubted performance hit is incurred only the first time a method is called, due to IL compilation, but it is compensated by "dead code" elimination [10] and various other optimization practices [11] which improve performance significantly.

Conclusions

Despite the advantages of the .NET framework we must have in mind that it is a commercial product aiming to profit and complying with certain limitations. However, the software development kit (SDK) [12] is free (including a detailed, well written documentation) and can be used without any restrictions for productive work. All .NET programming languages are expressive enough to address successfully the wide variety of problems and the miscellaneous philosophies exhibited by developers. The VMs seem to represent the next standard in programming as long as .NET and Java (which is the technological and commercial competitor) continue to evolve. No matter which programming approach one chooses, the following axiom is diachronic: "There does not now, nor will there ever, exist a programming language in which it is the least bit hard to write bad programs" [13].

Availability

The source code presented can be downloaded from: http://users.uoi.gr/nkourkou. For the "Language Interoperability" example, Salford Software FTN95 and Microsoft Visual Studio.NET are required [14]. Both are freely available by their manufacturers as trial versions. "Mapack" class library [15] has been used for some of the calculations.

References

[1] .NET Information (http://www.microsoft.com/net/)
[2] Mono Project open development initiative (http://www.go-mono.net)
[3] Jason Whittington, "Rotor, Shared Source CLI Provides Source Code for a FreeBSD Implementation of .NET", *MSDN Magazine*, **17**, 7 (2002)
[4] (i) P. Jane Brown, 11-14 (ii) Ralf W. Grosse-Kunstleve and Paul D. Adams 28-38 (iii) Juan Rodríguez-Carvajal and Javier González-Platas *Computing Commission Newsletter*, **1**, 50-58 (2003) (http://www.iucr.org/iucr-top/comm/ccom/newsletters/2003jan)
[5] PowDLL: a reusable XRPD .NET Component (http://www.ccp14.ac.uk/ccp/web-mirrors/powdll/nkourkou)
[6] Other DotNet Languages (http://c2.com/cgi/wiki?OtherDotNetLanguages)
[7] Gary Nutt, *Distributed Virtual Machines: Inside the Rotor CLI*, Addison-Wesley (2004).
[8] Cornell Theory Center (http://www.ctc-hpc.com/casestudies.html)
[9] Salford FTN95 and the .NET Framework Whitepaper (http://www.salfordsoftware.co.uk/compilers/support/documentation.html)
[10] Jeffrey Richter, *Applied Microsoft .NET Framework Programming*, Chapter 1, Microsoft Press (2002)
[11] Fahad Gilani, "C# In-Depth: Harness the Features of C# to Power Your Scientific Computing Projects", *MSDN Magazine*, **19**, 3 (2004)
[12] .NET Framework Software Development Kit (SDK) version 1.1 (http://msdn.microsoft.com/netframework/technologyinfo/howtoget/default.aspx)
[13] Lawrence Flon, *ACM SIGPLAN Notices*, **10**, 10 (1975)
[14] http://www.salfordsoftware.co.uk/compilers/ftn95 & http://msdn.microsoft.com/vstudio/
[15] Lutz Roeder, Mapack: a .NET class library for basic linear algebra computations (http://www.aisto.com/roeder/dotnet)

The ICR (Institute for Cancer Research) programs. Early crystallographic code implemented on the IBM 1620 in the beginning of the 1960's in the laboratory of A.L.(Lindo) Patterson

Dick van der Helm,
George Lynn Cross Research Professor(ret.), Department of Chemistry & Biochemistry, University of Oklahoma, Mailing address: 5895 Bay Pine Court, Ferndale, Washington State, 98248, USA - Email : dvdhelm@chemdept.chem.ou.edu ; WWW: http://cheminfo.chem.ou.edu/faculty/dvdh.html

In order to write about the programming of the IBM 1620 one has to set the stage about the use of crystallographic computing at that time. Generally in the forties and early fifties most structure solutions were still carried out in two dimensions. The results of two or three projections were combined to obtain a three dimensional picture of the structure of a molecule. Of course the power to use all data and to generate 3-dimensional Fouriers was realized from some remarkable structure solutions, such as Vitamin B12 and some others, but in general these calculations were very time consuming. For instance in the middle fifties I worked with Prof. Caroline MacGillavry on a new and active natural compound; a 3-d Fourier or S.F. calculation using punched cards and an IBM 604, took more than one month. This 604 was the main computer(!) of a large bank in Amsterdam and I used it during the night. In 1957 I joined the lab of Lynne Merritt at Indiana University, and here I had access to the IBM 650.As in many universities of that size the computing center consisted of just this one computer. It was quite expensive, and therefore beyond the financial reach of individual crystallographic laboratories. It was a computer with a rotating drum memory and four magnetic tape drives, all of which were used in the Fourier program which Lynne had written. Again I had the use of the whole computing center every night, and the art of the game was to keep all magnetic drives operating; several failures per night were common. However it gave me time to become more familiar with American literature. Besides the 650, the even more expensive and powerful 704 existed, but only the large oil companies, federal labs and large universities could afford this computer. At I.U. it was my job to write a general SFLS program which I presented in a meeting at the Mellon Institute in November 1958 (Acta Cryst.,12, 350).The way I treated symmetry was awkward, and at least in my mind, I was put to shame, in a lecture after me, by the elegant matrix and vector method for symmetry of space groups Bill Busing and Henry Levy described in the ORNL program for the 704. At least I learned a valuable lesson.

In 1959 I joined the laboratory of A.L. (Lindo) Patterson at the Institute for Cancer Research (ICR) in Philadelphia. The main project were the structures of the molecules involved in the citric acid cycle. I worked a lot with Jenny Glusker who had arrived several years prior after her beautiful work on B12. With the use of the 650's at Princeton, where my cousin was on the faculty, and Indiana University, we solved, together, the complete structure of sodium dihydrogen citrate from the 3-d Patterson (although it was called a vector synthesis in the lab),and refined the structure. Lindo immediately decided that we needed a computer in the lab which could do the 3-dimensional calculations, but of course there were financial constraints. The boundaries were clear: money and computing capability, memory and speed, and I was put in charge of choosing the computer. At that time there were one or two low-cost drum machines, but IBM had the design completed of a small and new random-access memory computer, the IBM 1620, and all the specs and program codes were already available. It was in the first wave of small computers. In order to decide if the computer would and could serve our purpose, I concentrated on the Fourier and SFLS calculations. The first one was complicated by the logic and the second one by the required algebra. It took a bout with the mumps to figure out that both could be done even for structures which were quite large small molecules.

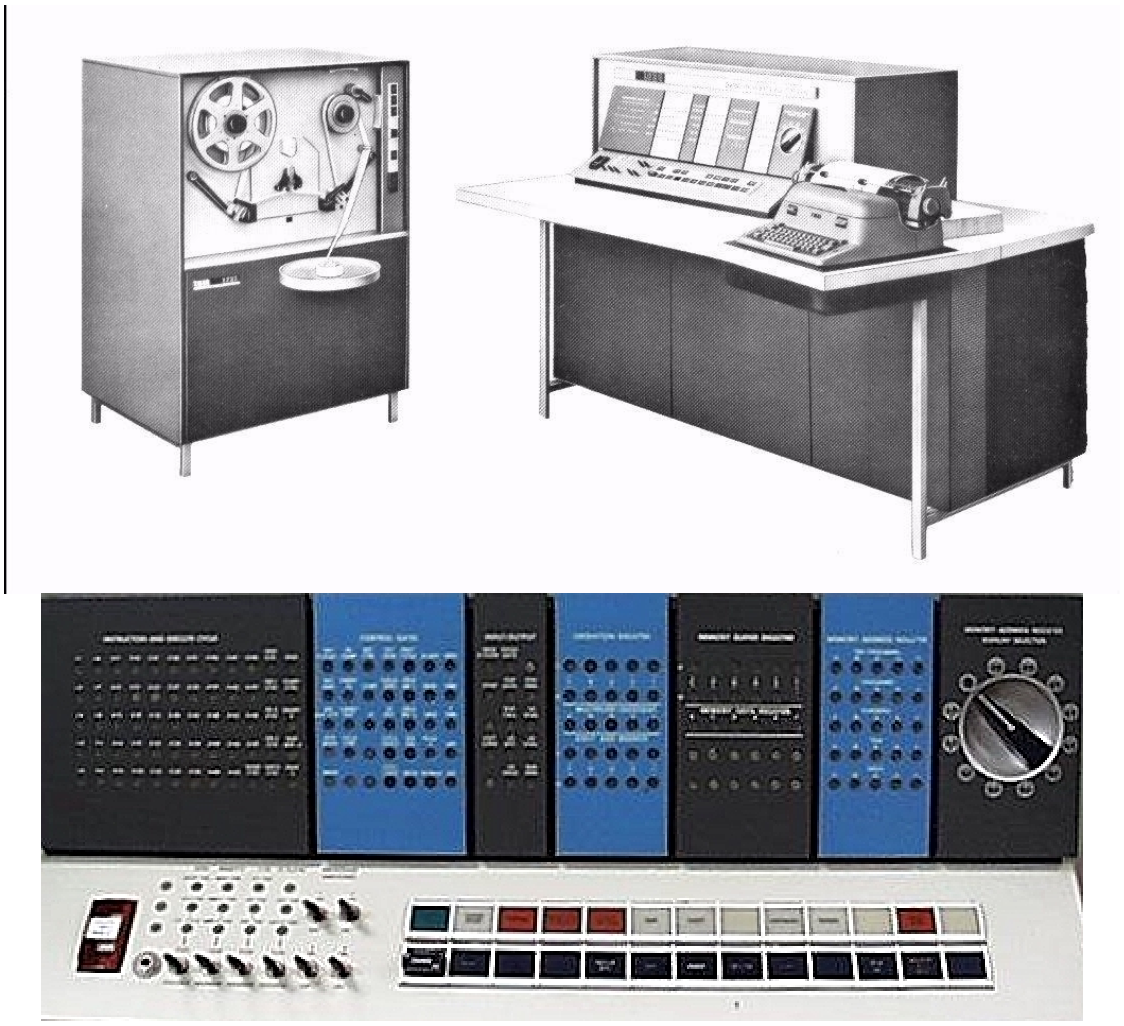


Fig.1: The basic IBM 1620 Data Processing System and Console.

The 1620 was arranged in two units(fig 1). One contained the computer, core storage and typewriter, the other the paper tape reader and punch. The construction of the 1620 was quite interesting. The whole memory was no more than 20000 decimal digits. Just imagine no Gbytes. The 3-d array of the memory is shown in fig 2 in which the bit core planes of the even and adjacent odd digit were combined. Each of the 20000 core digits could thus be read, written and checked, separately. The wordsize was variable, using the flag bit (F),while the check bit (C) was used for validity, although bit losses rarely happened. The computer could perform about 30 operations (arithmetic, data transmission, logical branch and input-output),while additional choices could be set with program switches. Each program instruction was 12 digits, two for the operation and five each for two addresses in memory(P and Q address). Addition, subtraction and multiplication operations were accomplished by a table-lookup method, which occupied 300 decimal digits in the core memory. Addition would be, for instance, the adding of the two numbers in the fields for which the P and Q address are the units digit, and then stored in the P address. Two decimal digits were required to represent an alphanumerical character.

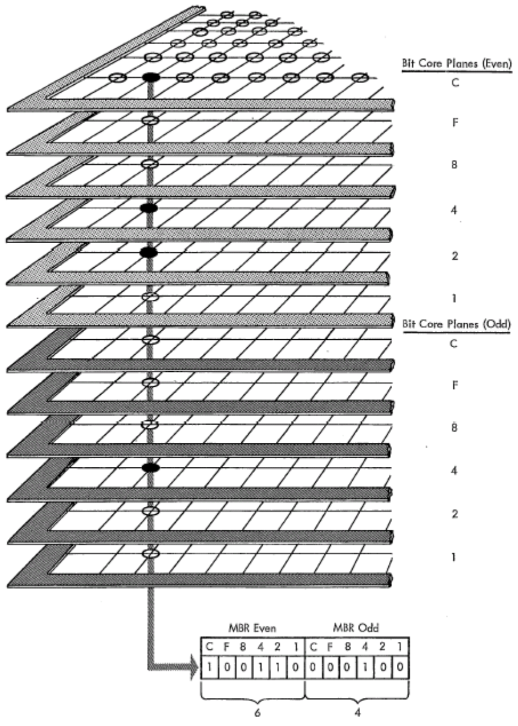


Fig 2: Schematic of the 3-d array of memory within an IBM 1620.

Fortran and utility programs were available but out of the question for the Fourier and SFLS programs due to the limited memory. Instead the programs had to be written in machine language or symbolic language (which allowed a one to one translation into machine language).

The Fourier program was written, initially, in machine language, which was not very smart, and later changed to symbolic language. The problems which had to be solved were logic, memory size, rounding and output, but the calculations were simple: addition and multiplication. Beevers and Lipson (see CompComm Newsletter number 2, Summer 2003) had already clearly shown that the number of operations could be reduced 10-15 fold by writing the summation in the, algebraically inelegant, multiplication form. This allowed as well to divide the summation into three separate ones. It was also clear that the limited memory would not allow the calculation to be done in one step. The results of the first summation easily would exceed the size of the memory. Rather than to punch these results out and sort, it was decided to read in the input tape of amplitudes (4 or 8 for each hkl, in the product form), once for each section. Also each x, y or z would be calculated at 1/100th intervals or multiples thereof. This is sufficient for even large small molecules. The sines and cosines were in a 100-entry table lookup. The answers of the first summation were stored at the end of the memory in such a manner that the sums in the beginning of this section of memory were used first in the subsequent second summation. The sums of the second summation were stored in the memory section between the program and the first summation answers, and they could therefore overlap, without interference, with the locations of the first summation answers. This in fact was the key to solve the memory problem. A separate program to prepare the input tape was written and it allowed sorting in such a way that the summation could be done in any desired sequence of x, y and z. It was up to the programmer to keep track of the decimal point in this fixed number machine, while still maintaining full four digit accuracy after the decimal point. Messages by the program indicated various places where overflow occurred or memory overlaps and a trial run would indicate where the problems occurred. These could be adjusted with the typed input parameters (shifts). The limitations of the program were not serious. If, for instance, the summation was over h and then k and then l, the product of the number of values along the b-axis multiplied by the maximum value of l had to be less than 1685. There were many different ways to output the results. The most convenient was one in which the cell dimensions as well as the angles between axes were approximated by horizontal, vertical and shift spacing while the numerical answers were translated into alphamerical and other characters with ranges of blanks, so that contouring by hand was not necessary.

This was especially useful with a 20-inch typewriter carriage, although this could be a dangerous instrument on its return motion. One section including input and output would take about 30 min. The program was rewritten for card input-output by G.S.D. King in Belgium.

In the SFLS program memory was also a problem. The available utility programs for exponentials, sines and cosines were slow, too accurate and took too much memory. New utility programs were written adapted to the 4-5 digit accuracy which is sufficient for the crystallographic calculations. For instance exp(-a.bcde) was calculated by 10-entry tables for exp(–a.0) and exp (–0.b),while exp–0.0cde was computed with a small Taylor series. The cosines and sines were reduced to the first quadrant, and calculated using : sin(0.abcd)=sin(0.ab)cos(0.00cd) + cos(0.ab)sin(.00cd), with table lookup for sin(0.ab) and cos(0.ab), and a small Taylor series for the rest. Still those two routines took about 1500 decimal digits. Normal calculator routines were adapted for a divide and a square root utility.

In those days (1960) there were still arguments among programmers if the formula's in the International Tables should be used or the ones I alluded to before, those based on P1 or P1 bar, with matrices and vectors for equivalent positions or even better for equivalent indices. The elegance of the latter method was evident in the ease by which contributions to the structure factor were calculated as well as the partial differentials for the L.S.sums. This certainly simplified the program code. The 1620 program was coded only for triclinic, monoclinic and orthorhombic, although it would not have been too difficult to include all space groups. The method also allowed a much more direct way to deal with anisotropic temperature factors. Due to the memory limitations it was a block diagonal matrix program (3 by 3 for positional and 6 by 6 for thermal parameters). The input was a data tape with indices, scattering factors and weight for all reflections, and a parameter tape with indices and coordinates. The limitations were most severe for orthorhombic space groups: 27 anisotropic atoms or 70 isotropic atoms, or a combination thereof, still quite reasonable for the type of structures solved in those days. The timing was between 10 and 25 sec per 10 atoms, which thus means 14 hrs for 20 anisotropic atoms and 1000 reflections. Being a fixed point computer, when coding was done in machine or symbolic language, there were as many as 20 distinct error messages which indicated where overflow occurred and the instruction booklet specified what particular action needed to be taken to eliminate the problem. The SFLS program started with a message: "And they all went…", and ended 10 or so hours later with: " …to the seashore". A quotation from a very pleasant Greek movie: "Never on Sunday". The Fourier program finished with the message: "Mooi", a dutch slang word which can mean all kinds of good things and is possibly best translated with the word: "Bien" in French.

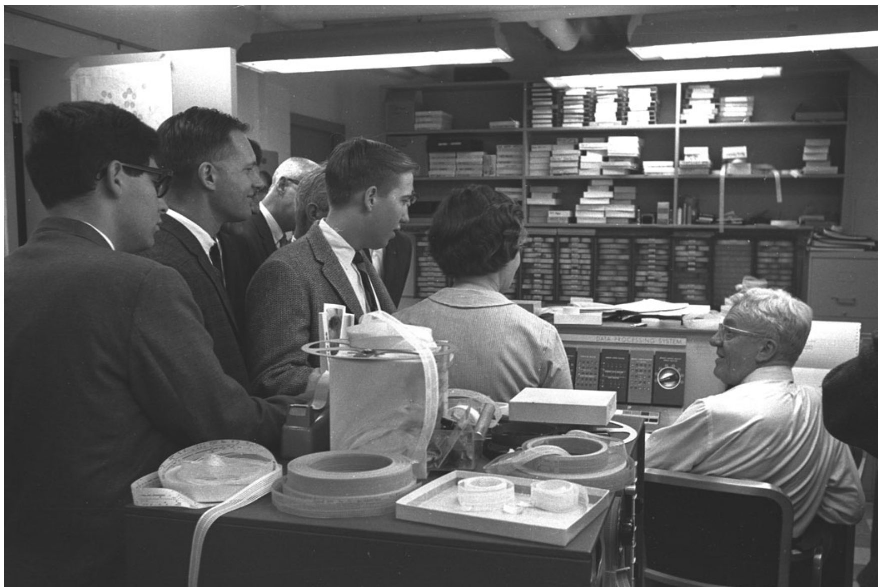


Fig 3: A.L.(Lindo) Patterson showing the IBM 1620 to a group of institute donors.

The Institute for Cancer Reasearch (ICR) was a private institution and the lease of the computer was a significant expense. Lindo therefore showed the computer to many many groups of donors (see Fig. 3). Lindo was very supportive to us making the programs, and wrote programs himself. The SFLS program, for instance, only calculated the least-squares sums, and Lindo wrote the program to calculate the coordinate shifts. He also wrote various programs on topology, because that was a major interest to him considering the question if the vector synthesis yielded a unique solution, but in addition topology intrigued him as a fundamental mathematical problem. Many other programs were written, especially by Carroll Johnson, see Fig 4. Examples are an absorption program and a goniostat program, and many are listed in one of his publications (Acta Cryst. 18, 1004 (1965)). When he moved to Oak Ridge he wrote the famous ORTEP program (not for the 1620). After Carroll and I moved to permanent positions Eric Gabe and Max Taylor joined the lab and wrote additional programs for the 1620. The programs, on paper tape, and instruction booklets were distributed for free, although I presume that donations were appreciated. I still have much of the correspondence up to 1962 and many friends were made. The programs went to Australia, New Zealand, UK and Europe, the USA and Canada.

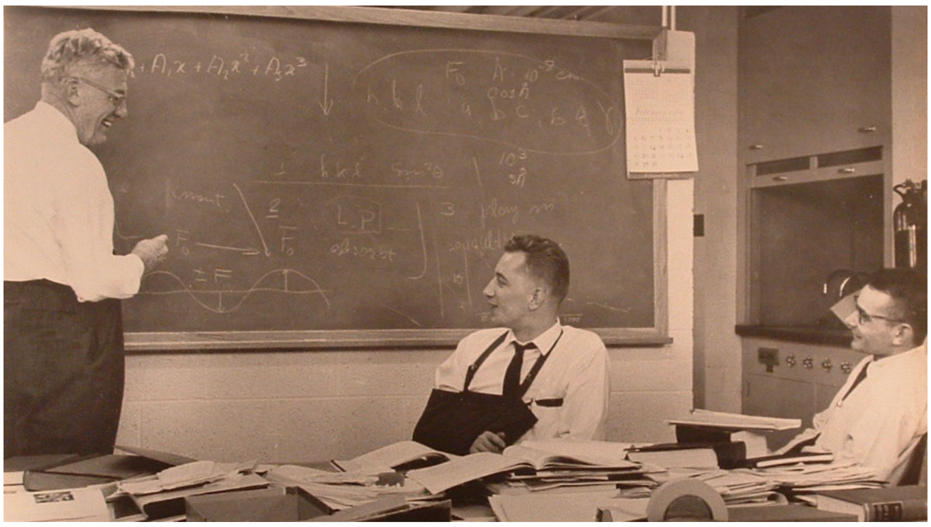


Fig 4: Left to Right - A.L.(Lindo) Patterson, the author and Carroll Johnson

I moved to the University of Oklahoma, which had a homemade large computer off-campus, and a 1620 on campus for general use. For a number of years I employed a technician who did run our programs at night, five days a week till the university acquired an IBM 360.

The following is included by request of the Editor:

One part of the Fourier program will be analyzed .It is the subroutine calculating from the sums of the first summation the sums to be used in the third summation, or in other words the second summation. It shows for instance the rounding of answers and also the shifting when overflows occurred in the making of the sums. The 1620 was a fixed point machine and the programmer had to keep track of the decimal point. The principle of the program was 4-digit accuracy. The program used a four digit cos or sin multiplied by a four digit amplitude giving an 8-digit answer. The leftmost 4 digits were used in the formation of sums, unless an overflow occurred which stopped the computer. One could find out if the overflow occurred in the first, second or third summation, and for each one could change the shift, normally thus 4, to for instance 5 (and even 6). It can also be set to 3 when one wanted more accuracy as in a difference Fourier. The shift parameters were a part of the input. A trial run would indicate a possible problem, although it did not occur often, and the program and parameter tapes had to be reloaded, with the indicated change in shift parameter.

In the Lipson and Cochran multiplication equation for the Fourier, there are 8 input amplitudes (noncentrosymmetric) or 4 input amplitudes (centrosymmetric) for each value of(h,k,*l*). The results of the first summation are respectively 4 or 2 amplitudes(sums) for each value (k,*l*,X). For the centrosymmetric case these are M(k,*l*,X) and N(k,*l*,X). The text described the fact that due to the limited memory only one section at a time could be calculated and therefore X is a definite value. The sums(amplitudes) resulting from the second summation in the centrosymmetric case are R(*l*,X,y), in which y varies from y(min) to y(max) with intervals of Δy. The third summation is thus assumed to be over *l*, from z(min) to z(max) with intervals of Δz. One could actually do the summation in any sequence by rearranging the amplitudes for the first summation on the input tape.

In the second summation the first amplitudes (k,*l,X)* which are used are M(0,0,X) and N(0,0,X), and their contributions to the sums, generally (*l,X,*y)*),* in this case R(0,X,y) are calculated for all values of y, and this is thus the part of the program shown below. Next, not part of the program code shown here, amplitudes (k,*l,X),* M(1,0,X) and N(1,0,X) are located and moved to the subroutine shown below, and their contributions for all values of y are added to the R(0,X,y) sums. This cycle continues till the last value of k, and consequently *l* changes for the M and N amplitudes. At that point the R(0,X,y) amplitudes are stored, not shown in the subroutine below. The next amplitudes are M(k,1,X) and N(k,1,X) in pairs and they are used with k from 0 to kmax, till *l* changes to 2, and the R(1,X,y) sums are stored, and so on.

As stated in the text the answers for the first summation are stored at the end of memory beginning with M(0,0,X) and N(0,0,X)…M(k,0,X) and N(k,0,X), and after that those for *l*=1 then *l*=2 etc, filling all the memory space till location 19479, which holds the amplitudes with maximum *l.* The answers of the second summation, however, are stored from location 6000 on. If overlap does occur during the second summation where R(*l*,X,y) sums could overlap with first summation amplitudes not yet used, an error message occurred (only rarely), and the only solution was to do the summation with a different sequence of h,k,and *l*.

Some details about instructions. Each basic instruction takes 12 digits. The first two digits are the operation, the next 5 the memory location of the P-field and the next 5 the memory location of the Q-field. A flag over the least significant digit means a negative number. In any other locations it means the left- end of the number .In other words numbers could be of any length with a minimum of 2 digits. The result of any multiplication was accumulated in location 99 and lower, with the length of the number being the sum of the size of the two numbers being multiplied. Add, subtract and multiply instructions are 21,22 and 23. The corresponding "immediate" instructions are 11, 12 and 13, where instead of the number specified in field Q, the actual value in field Q was used to add to the number in the P-field. Set flag and clear flag were 32 and 33. Transmission of a number in memory specified in the Q field to a memory location in the P field, was 26. A branch instruction (46),was an interrogation of one of 16 indicators, specified in Q8 and Q9 of the Q part of the instruction. Most commonly used were, for instance" equal zero" and " positive", indicators which were set by the previous arithmetic operation or compare instruction. Among the many and important branch instructions was also "branch on no flag", 44, and branch on digit, 43.

```
SUBROUTINE,MULTIPLICATION AND SUMMING FOR SECOND AND THIRD SUMMATION.

SUBR06 TF   COUNT1,NUMBR2          02370 26 05031 04900
       M    SUBR06-1,DELTA         02382 23 02369 05015
       SF   98                     02394 32 00098 00000
       TF   ADDIN,99               02406 26 05033 00099
       M    SUBR06-1,XMINT         02418 23 02369 05018
       SF   97                     02430 32 00097 00000
       TF   ARGUM,99               02442 26 04976 00099
SUBR04 BD   SUBR10,ARGUM-2         02454 43 02794 04974
```

```
SUBR09  MM ARGUM,9,10                02466 13 04976 000ô9
        A     SUBR01+11,99           02478 21 02501 00099
SUBR01  TR   491,5100,7              02490 31 00491 ô5100
        S     SUBR01 +11.99          02502 22 02501 00099
SUBR13  TFMCOUNT2,2,1                02514 16 05035 000ô2
        M    494,461                 02526 23 00494 00461
        TF   539,99                  02538 26 00539 00099
        M    498,465                 02550 23 00498 00465
        A     539,99                 02562 21 00539 00099
SUBR03  CF   532                     02574 33 00532 00000
SUBR11  SF   532                     02586 32 00532 00000
SUBR07  AM  536,5,10                 02598 11 00536 000ô5
        BNF SUBR02,539               02610 44 02634 00539
SUBR19  SF    535                    02622 32 00535 00000
SUBR02  A    6003,535,2              02634 21 ô6003 00535
        SM   COUNT2,1,10             02646 12 05035 000ô1
        BZ    SUBR14                 02658 46 02738 01200
```
(continue for noncentrosymmetric spacegroup ; go to 02738 for centrosymmetric space group)

```
SUBR14  AM   SUBR02+6,4,10           02738 11 02640 000ô4
        SM   COUNT1,1,10             02750 12 05031 000ô1
        BZ    SUBR05                 02762 46 02814 01200
        A      ARGUM,ADDIN           02774 21 04976 05033
        B      SUBR04                02786  49 02454
        DORG*-3(this allows the first two digits of the Q-field to be used for a constant, because every
digit saved helps with the little memory which the 1620 had, and it shifts the program sequence to 02794
rather than 02798)
 SUBR10  TDM ARGUM-2,0,11            02794 15 04974 0000ô
        B       SUBR09               02806 49 02466
        DORG*-3(see above)
SUBR05   BB                          02814 42
```
(the contributions for all values of y have been calculated and added, the program branches back to the place where a new set of amplitudes is retrieved for a new pass through this subroutine.

In the machine language program above, ô means a zero with a flag.

The two amplitudes M and N are in location 00461 (00458-00461) and 00465. The present value of k is stored in 02368 -02369, (SUBR06-1). The number of points along y is stored in 05031 (COUNT1). The kΔ y is calculated and stored in location 05033 (ADDIN), and ky in 04976 (ARGUM).

The next instruction BD,43, is kind of interesting. The idea is that an argument is allowed the be equal to 1.00 and larger (one might want to calculate y from 0.75 to 1.25). However in order to look up the appropriate cos and sin the arguments 1.00 and above were decreased by one whole cycle :f.i. 1.15 was set to 0.15. This is done in instructions at the end of the subroutine, instructions at 2794 and 2806. Setting the ARGUM to just two digits will not work because it could result in an overflow and the computer stops.

Instructions in 2466 and 2478 calculates the memory position for the cos-sin values for the particular ARGUM value, and the cos and sin values are stored in locations 491-498, each 4 digits long. Instruction in 2502 resets the instruction in 2490 for future use. The next few instruction calculate Mcosky+Nsinky, and stores it in location 532-539.

The next two instructions, CF 532 and SF 532 seem odd. Actually only the SF 532 instruction is necessary. It is part of the operations necessary to take care of possible overflow in making the sums and is initialized using the shift parameter for the second summation. The CF 532 does no harm. The result of the multiplications M(cosky) and N(sinky) is 8 digits long and normally the four most significant digits are used. However if in a trial run the second summation has overflow(s), one wants only the three most significant digits to be used. This will become clear in the next paragraph.

The next instruction is to round off the result in 532-539. Normally the 5 is added to the 5th significant digit(536). If the shift is 5 instead of the normal 4, this is initialized to (537), the 4th significant digit. Both positive and negative numbers are rounded properly, because the possible flag for a negative number in location 539 is not affected. Normally the answer in 532-535, a three digit number can be added safely to the four digit sum R(*l*,X,Y), instruction 2634, for which the memory location has previously been initialized. The coding as shown, however made this a 4 digit number by clearing the flag in 532 and setting it at 532 if the shift was 4 and at 531 if the shift was 5. If the result in 532-539 is negative, the flag has to be properly placed, normally on 535, but with a shift of 5 it is set in 534. This is done in instruction 2622.

The instructions 2670 till 2730 are for the case the spacegroup is noncentrosymmetric and two other first summation sums need to used for the calculation of S(*l*,Xy) values the code is not shown. Instructions 2738-2762 are a counter to check the number of values of y which has been done. If not all done the argument is increased by kΔy, and the program branches to instruction 2454 for the next value of ky. If all is done the program branches to SUBR05 to pick up new amplitudes.

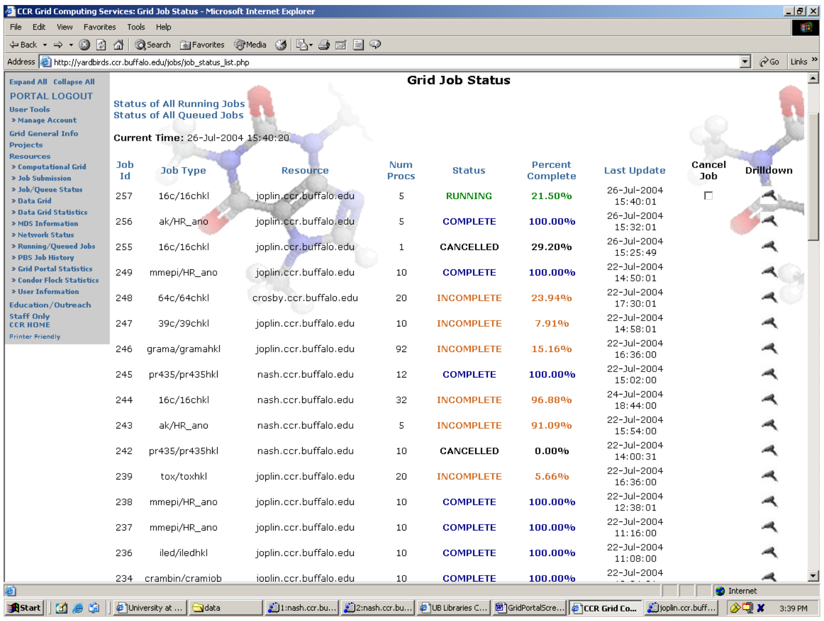
It is obvious from the small section of the program code which is shown here and that the program was written by an amateur, however the amateur was a responsible one, because the program worked properly. The code and flow diagrams for the programs are archived: http://www.ccp14.ac.uk/ccp/web-mirrors/ibm1620_xtal_code/. The flow diagrams for the Fourier program can be found under fourier synthesis , pages 43-48 or as images 41-46. The program code checking took quite a long time because writing in machine language any failure was your fault or you did not fully understand the meaning of the an instruction. Later on the program was rewritten for card input-output by G.S.D. King, who used many more self-explanatory mnemonic codes for constants, which made the program code much more easy to read, and also the flow sheet accompanying that program was professional. Also that program worked fine.

Meeting, workshop and school reports

During ACA 2003, July 17 - 22, 2004: Chicago, Illinois, USA: Report on ACA Chicago 2004 - 6.03: Advances in Computing Environments for Crystallography (http://www.hwi.buffalo.edu/ACA/ACA04/abstracts/S0603.html)

The Computing Environments session 6.03, sponsored by the ACA General Interest SIG, contained a good range of talks covering the use of the GRID, distributed computing, protein model building tools, through to visualisation and integration of single crystal 3D raw data and comparative visualization of molecular and protein structures.

The first talk by Russ Miller (miller@buffalo.edu ; co-authors: M.L. Green and C.M. Weeks) was about SnB (Shake-and-Bake)/BnP macromolecular structure solution and protein phasing software on the Grid (http://www.hwi.buffalo.edu/SnB/ http://www.ccr.buffalo.edu/grid/content/overview.htm). Russ clearly defined GRID computing and, in particular, what is not GRID computing, providing a breath of fresh air in laying down the law on the subject separating reality from hype and buzzwords. Pointing out that "The GRID" does not exist in the form that is commonly hyped and is currently under development, Russ elaborated on custom administrative tools written at the Center for Computational Research at SUNY-Buffalo in the context of the SUNY/Hauptman-Woodward Institute collaboration to make GRID computing practical for users and managers of GRID infrastructure. SnB/BnP is available via three existing GRID networks. Russ then gave a live Internet demonstration of submitting jobs to the SnB/BnP software via a standard web interface, followed by showing the tools that allowed users to check status of jobs, and quickly evaluate structure solution results. Software for collaborative examination and manipulation of molecular models via the Internet was also displayed. Queried during question time on whether authors of crystallographic software should make their programs GRID aware, Russ cautioned that unless you have a religious-type belief in the GRID, it might be too early to commit significant programming resources to GRID computing until the underlying systems management of GRID computing proves itself. Beta testers for the new SnB/BnP for GRID are most welcome and should contact Russ via the above address. The following screen-shot shows the status of submitted grid jobs of SnB being run directly on the ACDC Grid Portal via a Web Browser pointed to the Center for Computational Research at SUNY-Buffalo.



CCR Grid Computing Services: Grid Job Status - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://yardbirds.ccr.buffalo.edu/jobs/job_status_list.php
Go Links
Expand All Collapse All
PORTAL LOGOUT
User Tools
» Manage Account
Grid General Info
Projects
Resources
» Computational Grid
» Job Submission
» Job/Queue Status
» Data Grid
» Data Grid Statistics
» MDS Information
» Network Status
» Running/Queued Jobs
» PBS Job History
» Grid Portal Statistics
» Condor Flock Statistics
» User Information
Education/Outreach
Staff Only
CCR HOME
Printer Friendly
Internet
Start University at ... data 1:nash.ccr.bu... 2:nash.ccr.bu... UB Libraries C... GridPortalScre... CCR Grid Co... joplin.ccr.buff... 3:39 PM

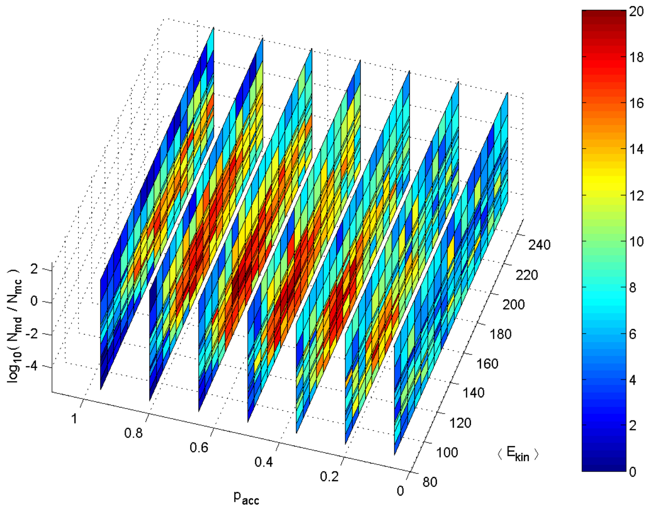
Grid Job Status

Status of All Running Jobs
Status of All Queued Jobs

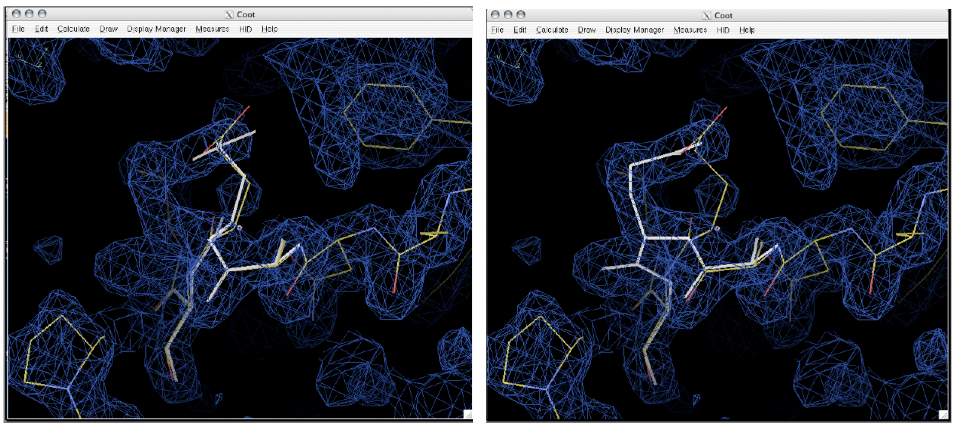
Current Time: 26-Jul-2004 15:40:20

Job Id	Job Type	Resource	Num Procs	Status	Percent Complete	Last Update	Cancel Job	Drilldown
257	16c/16chkl	joplin.ccr.buffalo.edu	5	RUNNING	21.50%	26-Jul-2004 15:40:01	☐	
256	ak/HR_ano	joplin.ccr.buffalo.edu	5	COMPLETE	100.00%	26-Jul-2004 15:32:01		
255	16c/16chkl	joplin.ccr.buffalo.edu	1	CANCELLED	29.20%	26-Jul-2004 15:25:49		
249	mmepi/HR_ano	joplin.ccr.buffalo.edu	10	COMPLETE	100.00%	22-Jul-2004 14:50:01		
248	64c/64chkl	crosby.ccr.buffalo.edu	20	INCOMPLETE	23.94%	22-Jul-2004 17:30:01		
247	39c/39chkl	joplin.ccr.buffalo.edu	10	INCOMPLETE	7.91%	22-Jul-2004 14:58:01		
246	grama/gramahkl	joplin.ccr.buffalo.edu	92	INCOMPLETE	15.16%	22-Jul-2004 16:36:00		
245	pr435/pr435hkl	nash.ccr.buffalo.edu	12	COMPLETE	100.00%	22-Jul-2004 15:02:00		
244	16c/16chkl	nash.ccr.buffalo.edu	32	INCOMPLETE	96.88%	24-Jul-2004 18:44:00		
243	ak/HR_ano	nash.ccr.buffalo.edu	5	INCOMPLETE	91.09%	22-Jul-2004 15:54:00		
242	pr435/pr435hkl	nash.ccr.buffalo.edu	10	CANCELLED	0.00%	22-Jul-2004 14:00:31		
239	tox/toxhkl	joplin.ccr.buffalo.edu	20	INCOMPLETE	5.66%	22-Jul-2004 16:36:00		
238	mmepi/HR_ano	joplin.ccr.buffalo.edu	10	COMPLETE	100.00%	22-Jul-2004 12:38:01		
237	mmepi/HR_ano	joplin.ccr.buffalo.edu	10	COMPLETE	100.00%	22-Jul-2004 11:16:00		
236	iled/iledhkl	joplin.ccr.buffalo.edu	10	COMPLETE	100.00%	22-Jul-2004 11:08:00		
234	crambin/cramiob	joplin.ccr.buffalo.edu	10	COMPLETE	100.00%	22-Jul-2004		

Anders Markvardsen (a.j.markvardsen@rl.ac.uk : co-authors: K. Shankland and W. David) of Rutherford-Appleton Laboratory in the UK, then discussed the use of distributed computing in the role of finding optimal Hybrid Monte-Carlo (HMC) parameters for structure solution from powder diffraction data. These optimal values can then be applied to structure solution software running on single workstations. Using distributed computing tools to link local workstations, results could be obtained in a couple of weeks that would have taken half a year or more using a single workstation. The following shows the plotted results for determining optimal HMC parameters based on the structure solution of Chlorothiazide.



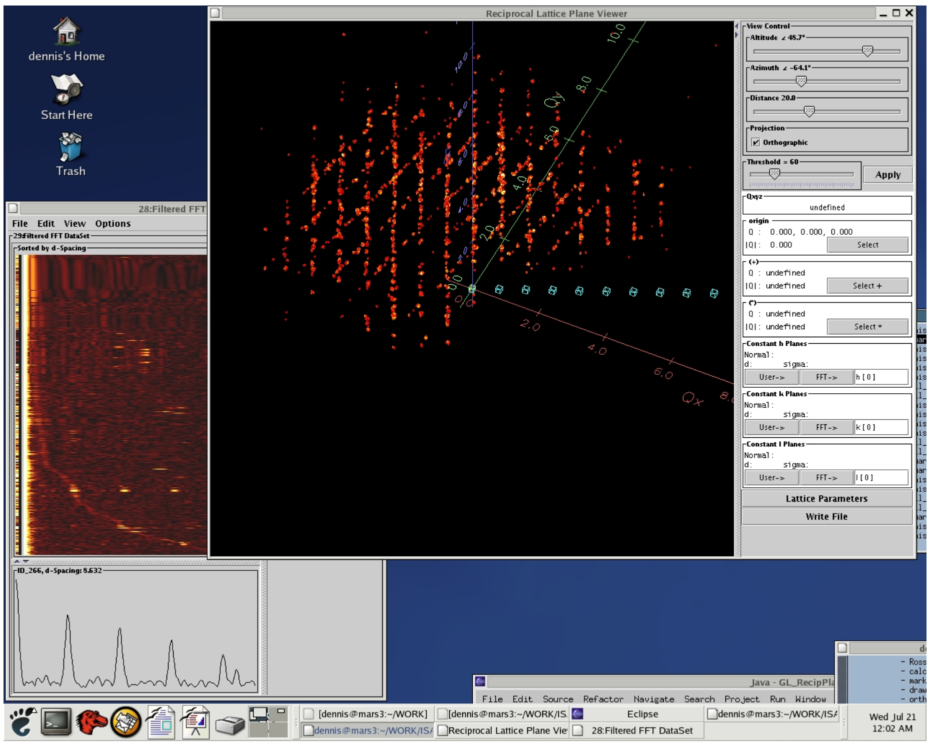
Showing a healthy disregard for the forces of computing conformity and conference requirements for MS-Windows compliance, Paul Emsley (emsley@ysbl.york.ac.uk) from York, UK used his Apple MacOS X laptop to demonstrate the COOT ((Crystallographic Object)-Oriented Toolkit) Model Building Tools for protein crystallography. Coot is part of the CCP4 Molecular Graphics Project and has some features that resemble those of Frodo, O, Quanta and XtalView's XFIT. Paul's live demonstration showing COOT re-optimising the incorrect orientation of a residue in real-time drew "ooh's" and "ahh's" from the audience. The two screen images below show the before and after of this demonstration. COOT (http://www.ysbl.york.ac.uk/~emsley/coot/) is freely available in source code form under the GNU GPL Licence, and compiled binaries for a variety of operating systems (SGI IRIX, Mac OS X, Redhat Linux) are available via (http://www.ysbl.york.ac.uk/~emsley/software/binaries/).



Coot
File Edit Calculate Draw Display Manager Measures HID Help
Coot
File Edit Calculate Draw Display Manager Measures HID Help

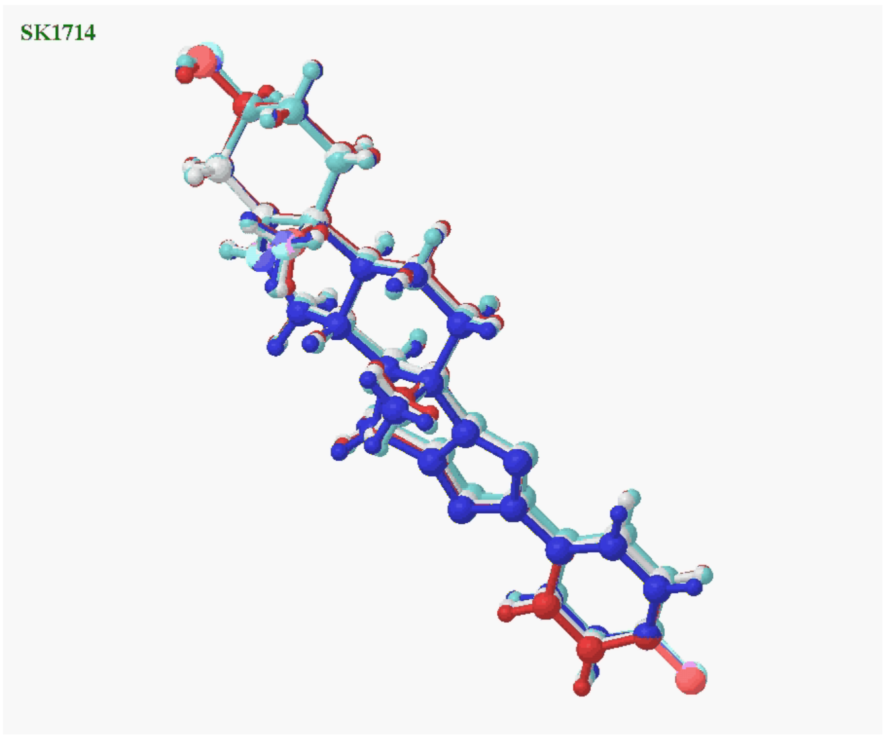
Dennis Mikkelson (mikkelsond@uwstout.edu : co authors: A. Schultz, P. Peterson, R. Mikkelson, T. Worlton, J. Hammonds, J. Cowan, Martha Miller, C. Bouzek, Michael Miller), a senior computer scientist at University of Wisconsin-Stout introduced a GPL'd user friendly software package for viewing raw neutron Time-of-Flight (TOF) single crystal data, with the option of indexing and integration (http://www.pns.anl.gov/computing/isaw/). Speeds of visualisation for reconstructed raw image files of reciprocal space collected with multiple detectors were stated as being performed with a second or so. Both manual and computer based indexing options were shown for handling multiple crystallites; and the software has the ability to perform integration of 3D diffraction spots. Various "wizards" to aid in analysis, and the hkl slice viewer are in the latest build (1.7.1 alpha 7) available on the above ISAW

(Integrated Spectral Analysis Workbench software) website. The 3D reciprocal lattice view described in the talk is expected to be in the 1.7.1 "final" build by late August. Under the tyrannical direction of the session chair, Dennis quickly flicked through remaining slides to remain on time. Had the chair of the session (this humble scribe) been more on the ball and quick witted, he would have seen the error of keeping this talk to time and insisted that Dennis elaborate on a slide describing the future of this software, which included an invitation for collaborators to help develop the software. This software seems to represent not only an opportunity to the TOF single crystal community, but also X-ray CCD based crystallographers in providing the freedom to interact and integrate their raw single crystal data in a highly flexible manner. Free software aficionados will note the GPL definition of the word "freedom" is being used here. The following shows the software being manipulated both manually (and with computer controlled algorithms) to assist in sorting out effects of multiple single crystals prior to indexing, cell assignments and integration.



View Control
Altitude ∠ 48.7°
Azimuth ∠ -64.1°
Distance 20.0
Projection
☑ Orthographic
Threshold = 60 Apply
Qxyz: undefined
origin
Q : 0.000, 0.000, 0.000
|Q| : 0.000 Select
(+)
Q : undefined
|Q| : undefined Select +
(*)
Q : undefined
|Q| : undefined Select *
Constant h Planes
Normal:
d: sigma:
User-> FFT-> h [0]
Constant k Planes
Normal:
d: sigma:
User-> FFT-> k [0]
Constant l Planes
Normal:
d: sigma:
User-> FFT-> l [0]
Lattice Parameters
Write File

The final talk of the session was that of David Duchamp (djduchamp@aol.com), showing the latest feature of CrystMol (Mac and MS Windows - http://www.crystmol.com/) for visually comparing potentially similar molecules from different structure files, or within the same structure where Z' is greater than 1; as well as proteins. People comparing polymorphs, or a chemically similar series of structures, could find this very beneficial and time saving. Molecules can be compared automatically; using a point and click menu; or via the CrystMol scripting system. RMS differences are also listed. Following is an example of CrystMol comparing the Z'=4 structure from S.Thamotharan, V. Parthasarathi, R. Gupta, D.P. Jindal and A. Linden (2004), Acta Cryst C60, o405-o407.



Besides thanking the speakers for their presentations, thanks must also go to the staff of the Hyatt-Regency, Chicago for their effective assistance in the set up of presenter laptops.

Lachlan Cranswick

Call for Contributions to the Next CompComm Newsletter

The third issue of the Compcomm Newsletter is expected to appear around January of 2005 with the primary theme of "At Right Angles to Conventional Crystallographic reality: incommensurate, quasicrystals, pair distribution functions and magnetic structures". Articles related to the control and visualisation of raw single crystal image data for the elucidation of many of the above types of structural problems is also very welcome and appropriate. If no-one is else can be co-opted, the newsletter will be edited by Lachlan Cranswick.

Contributions would be also greatly appreciated on matters of general interest to the crystallographic computing community, e.g. meeting reports, future meetings, developments in software, algorithms, coding, programming languages, techniques and other news.

Please send articles and suggestions directly to the editor.

Lachlan M. D. Cranswick
NPMR, NRC,
Building 459, Station 18,
Chalk River Laboratories,
Chalk River, Ontario,
Canada, K0J 1J0
E-mail: lachlan.cranswick@nrc.gc.ca
WWW: http://neutron.nrc.gc.ca/peep.html#cranswick