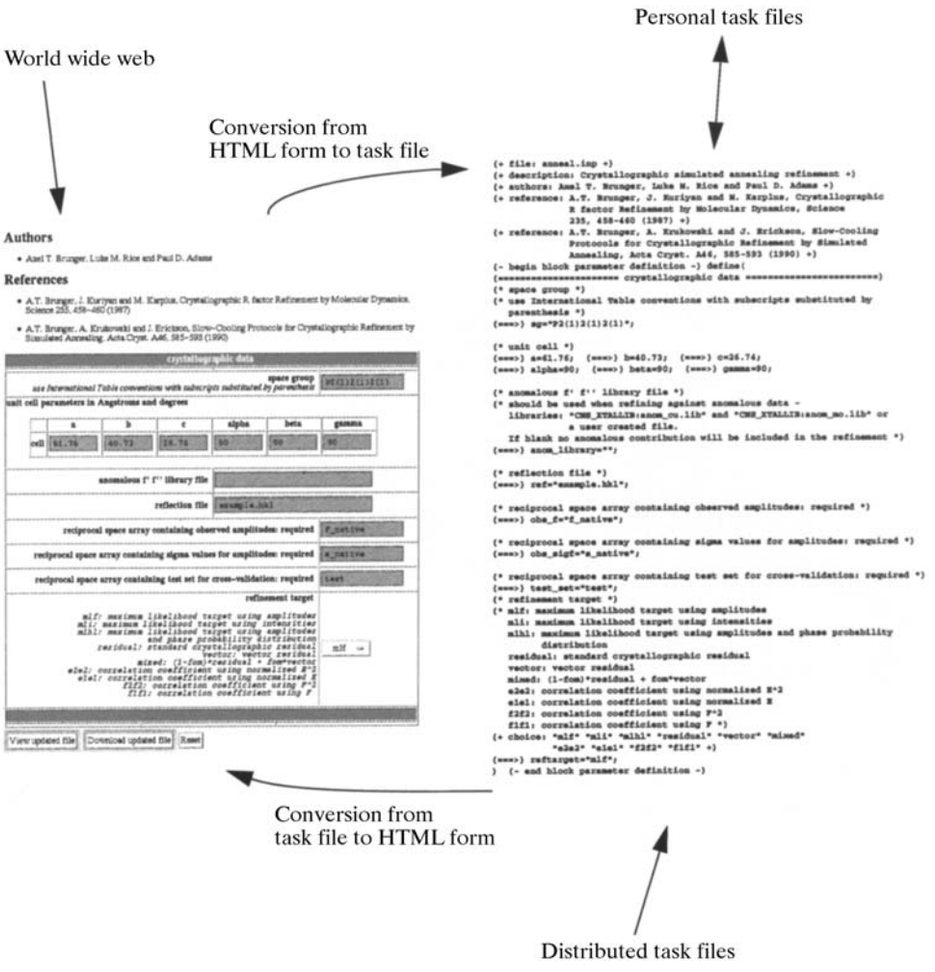


Fig. 25.2.3.10. Use of the CNS HTML form page interface, emphasizing the correspondence between input fields in the form page and parameters in the task file.

validated $\sigma_A$ estimates, determines the weighting scheme between the X-ray refinement target function and the geometric energy function (Brünger *et al.*, 1989), refines a flat bulk solvent model (Jiang & Brünger, 1994) and an overall anisotropic $B$ value for the model by least-squares minimization, and subsequently refines the atomic positions by simulated annealing. Options are available for specification of alternate conformations, multiple conformers (Burling & Brünger, 1994), noncrystallographic symmetry constraints and restraints (Weis *et al.*, 1990), and 'flat' solvent models (Jiang & Brünger, 1994). Available target functions include the maximum-likelihood functions MLF, MLI and MLHL (Pannu & Read, 1996*a*; Adams *et al.*, 1997; Pannu *et al.*, 1998). The user can choose between slow cooling (Brünger *et al.*, 1990) and constant-temperature simulated annealing, and the respective rate of cooling and length of the annealing scheme. For a review of simulated annealing in X-ray crystallography, see Brünger *et al.* (1997).

During simulated-annealing refinement, the model can be significantly improved. Therefore, it becomes important to recalculate the cross-validated $\sigma_A$ error estimates (Kleywegt & Brunger, 1996; Read, 1997) and the weight between the X-ray diffraction target function and the geometric energy function in the course of the refinement (Adams *et al.*, 1997). This is important for the maximum-likelihood target functions that depend on the cross-validated $\sigma_A$ error estimates. In the simulated-annealing task file, the recalculation of $\sigma_A$ values and subsequently the weight for the crystallographic energy term are carried out after initial energy minimization, and also after molecular-dynamics simulated annealing.

#### 25.2.3.10. *Conclusions*

CNS is a general system for structure determination by X-ray crystallography and solution NMR. It covers the whole spectrum of methods used to solve X-ray or solution NMR structures. The multi-layer architecture allows use of the system with different levels of expertise. The HTML interface allows the novice to perform standard tasks. The interface provides a convenient means of editing complicated task files, even for the expert (Fig. 25.2.3.10). This graphical interface makes it less likely that an important parameter will be overlooked when editing the file. In addition, the graphical interface can be used with any task file, not just the standard distributed ones. HTML-based documentation and graphical output is planned in the future.

Most operations within a crystallographic algorithm are defined through modules and task files. This allows for the development of new algorithms and for existing algorithms to be precisely defined and easily modified without the need for source-code modifications.

The hierarchical structure of CNS allows extensive testing at each level. For example, once the source code and CNS basic commands have been tested, testing of the modules and task files is performed. A test suite consisting of more than a hundred test cases is frequently evaluated during CNS development in order to detect and correct programming errors. Furthermore, this suite is run on several hardware platforms in order to detect any machine-specific errors. This testing scheme makes CNS highly reliable.

Algorithms can be readily understood by inspecting the modules or task files. This self-documenting feature of the modules provides a powerful teaching tool. Users can easily interpret an algorithm and compare it with published methods in the literature. To our knowledge, CNS is the only system that enables one to define symbolically any target function for a broad range of applications, from heavy-atom phasing or molecular-replacement searches to atomic resolution refinement.

### 25.2.4. The *TNT* refinement package

(D. E. Tronrud and L. F. Ten Eyck)

#### 25.2.4.1. *Scope and function of the package*

*TNT* (Tronrud *et al.*, 1987) is a computer program package that optimizes the parameters of a molecular model given a set of observations and indicates the location of errors that it cannot correct. Its authors presume the principal set of observations to be the structure factors observed in a single-crystal diffraction experiment. To complement such a data set, which for most macromolecules has limitations, stereochemical restraints such as standard bond lengths and angles are also used as observations.

A molecule is parameterized as a set of atoms, each with a position in space, an isotropic $B$ factor and an occupancy. The

complete model also includes an overall scale factor, which converts the arbitrary units of the measured structure factors to e $\text{Å}^{-3}$, and a two-parameter model of the electron density of the bulk solvent.

Because a *TNT* model of a macromolecule does not allow anisotropic *B* factors, *TNT* cannot be used to finish the refinement of any structure that diffracts to high enough resolution to justify the use of these parameters. If one has a crystal that diffracts to 1.4 Å or better, the final model should probably include these parameters and *TNT* cannot be used. One may still use *TNT* in the early stages of such a refinement because one usually begins with only isotropic *B*'s.

At the other extreme of resolution, *TNT* begins to break down with data sets limited to only about 3.5 Å data. This breakdown occurs for two reasons. First, at 3.5 Å resolution, the maps can no longer resolve $\beta$-sheet strands or $\alpha$-helices. The refinement of a model against data of such low resolution requires strong restraints on dihedral angles and hydrogen bonds – tasks for which *TNT* is not well suited. Second, the errors in an initial model constructed with only 3.5 Å data are usually of such a magnitude and quality that the function minimizer in *TNT* cannot correct them.

#### 25.2.4.2. *Historical context*

The design of *TNT* began in the late 1970s, and the first publishable models were generated by *TNT* in 1981 (Holmes & Matthews, 1981). Its design was greatly influenced by observations of the strength and weaknesses of programs then available.

The first refinement of a protein model was performed by Jensen and co-workers at the University of Washington (Watenpaugh *et al.*, 1973). This structure refinement was atypical because of the availability of high-resolution data. The techniques of pre-least-squares small-molecule refinement were simply applied to this much larger model. Since many of the calculations were performed manually, no comprehensive software package was created for distribution.

It was quickly realized that for macromolecular refinement to become common, the calculations had to be fully automated and ideal stereochemistry had to be enforced. In the late 1970s, four programs became available, all of which automated the refinement calculations, but each implemented the enforcement of stereochemistry in different ways. They were *PROLSQ* (Hendrickson & Konnert, 1980), *EREF* (Jack & Levitt, 1978), *CORELS* (Sussman *et al.*, 1977) and *FFTSF* (Agarwal, 1978). *PROLSQ* was, ultimately, the most popular.

At one end of the spectrum lay *FFTSF*. This program optimized its models to the diffraction data while completely ignoring ideal geometry. Following a number of iterations of optimizing the fit of the model to the structure factors, the geometry was idealized by running a separate program. At the other extreme was *CORELS*. It optimized its models to the diffraction data while allowing no deviations from ideal stereochemistry. The model was allowed to change only through the rotation of single bonds and the movement of rigid groups. Both approaches were frustrating to a certain extent. With *FFTSF* it was a struggle to find a model that agreed with all observations at once. With *CORELS* it was difficult to get the model to fit the density, because small and, apparently, insignificant deviations from ideality often added up after many residues to large and significant displacements, and these were forbidden. Neither approach to stereochemistry seemed very convenient, although *CORELS* was used for early-stage refinement for many years because of its exceptional radius of convergence.

Both *PROLSQ* and *EREF* enforced ideal stereochemistry and agreement with the diffraction data simultaneously. This strategy proved very convenient and generated models that satisfied their users. The two programs differed significantly in the form in which they required the ideal values be entered. *PROLSQ* required that the ideal values for both bond lengths and bond angles be entered as distances, *e.g.* an angle was defined by the distance between the two extreme atoms. *EREF* required that the standard value for an angle simply be entered as the number of degrees. Since *EREF* stored its library of standard values in the same terms as those with which people were familiar, it was much easier to enter the values.

These two programs differed in another way as well. *PROLSQ* stored ideal values for the stereochemistry of each type of residue (*e.g.* alanine, glycine *etc.*), while *EREF* parameterized the library in terms of atom types. For example, the angle formed by three atoms, the first a keto oxygen, the second a carbonyl carbon and the third an amide nitrogen, would have a particular ideal value regardless of where these three atoms occurred. In this matter, *PROLSQ* was more similar to the thought patterns of crystallographers.

#### 25.2.4.3. *Design principles*

*TNT* was designed with three fundamental principles in mind. Each principle has a number of consequences that shaped the ultimate form of the package.

##### 25.2.4.3.1. *Refinement should be simple to run*

The user should not be burdened with the choice of input parameters that they may not be qualified to choose. They also should not be forced to construct an input file that is obscure and difficult to understand. It is hard now to remember what most computer programs were like in the 1970s. Usually, the input to the program was a block of numbers and flags where the meaning of each item was defined by its line and column numbers. This block not only contained information the programmer could never anticipate, like the cell constants, but defined how the computer's memory should be allocated and obscure parameters that could only be estimated after careful reading of research papers.

*TNT* was one of the first programs in crystallography to have its input introduced with keywords and to allow input statements to come in any order. As an example of the difference, consider the resolution limits. Usually, a crystallographic program would have a line in its input similar to

```
99.0, 1.9,
```

One had to recognize this line amongst many as the line containing the resolution limits. (In many programs, a value of 99 was used to indicate that no lower-resolution limit was to be applied.) In *TNT* the same data would be entered as

```
RESOLUTION 1.9
```

The keyword identifies the data as the resolution limit(s). If the statement contained two numbers, they were considered the upper and lower limits of the diffraction data.

The preceding example also shows how default values can be implemented by a program much more safely with keyword-based input. In the previous scheme, if a value was ever to be changed by the user, its place had to be allocated in the input block. This often left numbers floating in the block which were almost never changed, and because they were so infrequently referred to, they were usually unrecognized by the user. It was quite possible for one of these numbers to be accidentally changed and the error unnoticed for quite some time. When the data are introduced with keywords, a data item is not mentioned if the default value is suitable.

##### 25.2.4.3.2. *Refinement should run quickly and use as little memory as possible*

The most time-consuming calculations in refinement are the calculation of structure factors from atomic coordinates and the

calculation of derivatives of the part of the residual dependent upon the diffraction data with respect to the atomic parameters. The quickest means of performing these calculations requires the use of space-group-optimized fast Fourier transforms (FFTs). The initial implementation of *TNT* used FFTs to calculate structure factors, but the much slower direct summation method to calculate the derivatives. Within a few years, Agarwal's method (Agarwal, 1978; Agarwal *et al.*, 1981) was incorporated into *TNT* and from then on all crystallographic calculations were performed with FFTs.

The FFT programs of Ten Eyck (1973, 1977) made very efficient use of computer memory. Another means of saving memory was to recognize that the code for calculating stereochemical restraints did not need to be in the memory when the crystallographic calculations were being performed and *vice versa*. There were two ways to save memory using this information. One could create a series of 'overlays' or one could break the calculation into a series of separate programs. The means for defining an overlay structure were never standardized and could not be ported from one type of computer to another and were, therefore, never attempted in *TNT*. For this reason, and a number of others mentioned here, *TNT* is not a single program but a collection of programs, each with a well defined and specialized purpose.

#### 25.2.4.3.3. *The source code should not require customization for each project*

The need to state this goal seems remarkable in these modern times, but the truth is that most computer programs in the 1970s required specific customizations before they could be used. The simplest modifications were the definitions of the maximum number of atoms, residues, atom types *etc.* accepted by the program. These modifications are still required in Fortran77 programs because that language does not allow the dynamic allocation of memory. However, in most programs today the limits are set high enough that the standard configuration does not present a problem.

The most difficult modification required for programs like *PROLSQ* was to adapt the calculations to the space group in hand. Their authors usually included code for the space groups they were particularly interested in, leaving all others to be implemented by the user. Writing code for a new space group was often a daunting task for someone who was not an expert programmer and had no tools for testing the modifications.

It is too burdensome to require the user to understand sufficiently the internal workings of a complex calculation that they can code and debug central subroutines of a refinement program. In its initial implementation, *TNT* avoided this problem, to an extent, by performing the space-group-specific calculations in separate programs. At least the user did not need to modify an existing program. All that was required was the construction of a program that read the proper format file, performed the calculation and wrote its answer in the proper format. The user was required to supply both a program that could calculate structure factors from the model and another program that could calculate the derivative of the diffraction component of the residual function with respect to the atomic parameters of the model.

While a structure-factor program could usually be located, either by finding an existing program or by expanding the model to a lower-symmetry space group for which a program did exist, the requirement of creating a derivative program proved too great a burden. The derivation of the space-group-specific calculation, its implementation and debugging proved too difficult for almost everyone, and this design was quickly abandoned. Instead, an implementation of Agarwal's (1978) algorithm was created. In this method, the derivatives are calculated with a series of convolutions with an $F_o - F_c$ map. The calculation of the map is the only space-group-specific part of the calculation, and this was done with a separate program for calculating Fourier syntheses. Such programs were as easy to come by as structure-factor calculation programs and could be replaced by a lower-symmetry program if required.

While it is easier to find or write a program that only calculates a Fourier transform and much easier to debug one than to debug a modification to a larger and more complex program, it is still difficult. The lack of availability of programs for the space group of a crystal often prevented the use of *TNT*. Over time, programs for more space groups were written and distributed with *TNT*. Eventually, a method was developed by one of *TNT*'s authors in which FFTs could be calculated using a single program as efficiently as the original space-group-specific programs. Once this program existed, there was no longer the need for isolated structure-factor and Fourier synthesis programs. These calculations have disappeared into the heart of *TNT*, and *TNT* consists of many fewer programs today than in the past.

### 25.2.4.4. *Current structure of the package*

*TNT* presents different faces to different users. Some users simply want to run refinement; they see the shell interface. Others want to use the *TNT* programs in untraditional ways; they see the program interface. A few users want to change the basic calculations of *TNT*; they see the library interface.

The shell interface is the view of *TNT* that most people see. It is the most recent structural addition, having been added in release 5E in 1995. At this level, the restraints, weights and parameters of the model are described in the '*TNT* control file' and the user performs particular calculations by giving commands at the shell prompt. For example, refinement is performed with the 'tnt' command and maps are calculated for examination with some graphics program with the 'make_maps' command. *TNT* is supplied with about two dozen shell commands. These commands allow the running of refinement, the conversion of the model to and from *TNT*'s internal format, and the examination of the model to locate potential problem spots. The *TNT Users' Guide* describes the use of *TNT* at this level.

The program interface consists of the individual *TNT* programs along with their individual capabilities. *TNT* consists of the program *Shift*, which handles all the minimization calculations, a program for each module (restraints that fall into a common class, *e.g.* diffraction data, ideal stereochemistry and noncrystallographic symmetry) and a number of utility programs of which the most important member is the program *Convert*, which reads and writes coordinate files in many formats. The user can write shell scripts (or modify those supplied with *TNT*) to perform a great many tasks that cannot be accessed with the standard set of scripts. The *TNT Reference Manual* describes the operation of each program.

If the programs in *TNT* do not perform the calculation wanted, the source code can be modified. The source code to *TNT* is supplied with the standard distribution. In order to make the code more manageable and understandable, it is divided into half a dozen libraries. All *TNT* programs use the lowest-level library to ensure consistency of the 'look and feel' and use the basic data structures for storage of the model's parameters and the vital crystal data. To add new functionality, one can either modify an existing program, write a new program using the *TNT* libraries as a start, or write a new program from scratch ignoring the *TNT* libraries. As long as a program can read and write files of the same format as the rest of *TNT*, it will work well with *TNT*, even if it does not share any code. A library exists, but is not copyrighted, that contains subroutines to read and write the crystallographic file formats used by the rest of *TNT*.

### 25.2.4.5. *Innovations first introduced in TNT*

*TNT* was not only designed to be an easy-to-use tool for the refinement of macromolecular models, but also as a tool for testing

new ideas in refinement. Since its source code is designed to allow easy reordering of tasks and simple modifications, a number of innovations in refinement made their first appearance in *TNT*. These features include the following.

#### 25.2.4.5.1. *Identifying and restraining symmetry-related contacts (1982)*

Without a search for symmetry-related bad contacts, it was quite common to build atoms into the same density from two different sides of the molecule. A number of models in the PDB contain these types of errors because neither the refinement nor the graphics programs available at that time would indicate this type of error.

#### 25.2.4.5.2. *The ability of a single package to perform both individual atom and rigid-body refinement (1982)*

Prior to *TNT*, one often started a refinement with rigid-body refinement using *CORELS* and then switched to another program. *TNT* was the first refinement package to allow both styles of refinement. One was not required to learn about two different packages when running *TNT*.

#### 25.2.4.5.3. *Space-group optimized FFTs for all space groups (1989)*

This innovation allowed *TNT* to run efficiently in all space groups available to macromolecular crystals.

#### 25.2.4.5.4. *Modelling bulk solvent scattering via local scaling (∼1989)*

With a simple and quick model of the scattering of the bulk solvent in the crystal (Tronrud, 1997), the low-resolution data could be used in refinement for the first time. The inclusion of these data in the calculation of maps greatly improved their appearance.

#### 25.2.4.5.5. *Preconditioned conjugate-gradient minimization (1990)*

This method of minimization (Axelsson & Barker, 1984; Tronrud, 1992) allows the direct inclusion of the diagonal elements of the second-derivative matrix and the indirect inclusion of its off-diagonal elements. An additional benefit is that it allows both positional parameters and *B* factors to be optimized in each cycle. Previously, one was required to hold one class of parameter fixed while the other was optimized. It is much more efficient and simpler for the user to optimize all parameters at once. This method, because it incorporates the diagonal elements directly, produces sets of *B* factors that agree with the diffraction data better than those from the simple conjugate-gradient method.

#### 25.2.4.5.6. *Restraining stereochemistry of chemical links to symmetry-related molecules (∼1992)*

It is not uncommon for crystallization enhancers to be found on a special position in the crystal. In addition, cross-linking the molecules in a crystal is often done for various reasons. In both cases, the model contains chemical bonds to a molecule or atoms in another asymmetric unit of the crystal. In order for the stereochemistry of these links to be properly restrained, it must be possible to describe such a link to the refinement program.

#### 25.2.4.5.7. *Knowledge-based B-factor restraints (∼1994)*

When the resolution of the diffraction data set is less than about 2 Å, the individual *B* factors of a refined model are observed to vary wildly from atom to atom, even when the atoms are bonded to one another. This pattern is not reasonable if one interprets the *B* factor as a measure of the vibrational motion of the atom. Traditionally, one applies an additional restraint on the *B* factors of the model, where the ideal value for the difference in *B* factor for two bonded atoms is zero.

Since it is clear from examinations of higher-resolution models that the *B* factors generally increase from one side of a bond to the other (*e.g.* moving from the main chain to the end of a side chain), the traditional restraint is flawed. A restraint library was generated (Tronrud, 1996) where each bond in a residue is assigned a preferred increment in *B* factor and a confidence (standard deviation) in that increment.

#### 25.2.4.5.8. *Block-diagonal preconditioned conjugate-gradient minimization with pseudoinverses (1998)*

With this enhancement, *TNT*'s minimizer treats the second-derivative matrix as a collection of $5 \times 5$ element blocks along its diagonal, one block for each atom. While this method improves the rate of convergence for noncrystallographic symmetry restraints, its most significant feature is that it allows the refinement of atoms located on special positions without special handling by the user.

#### 25.2.4.5.9. *Generalization of noncrystallographic symmetry operators to include shifts in the average B factor (1998)*

It is rather common in crystals containing multiple copies of a molecule in the asymmetric unit for one or more molecules to have a higher *B* factor than the others. If the transformation that generates each copy of the molecule consists only of a rotation and translation of the positions of the atoms, the difference in *B* factors cannot be modelled. The transformations used in *TNT* now consist of a rotation, translation, a *B*-factor shift and an occupancy shift.

### 25.2.4.6. *TNT as a research tool*

*TNT* was intended not only as a tool for performing refinement, but as a tool for developing new ideas in refinement. While most of the latter has been done by *TNT*'s authors, several others have made good use of *TNT* in this fashion. If one has an idea to test, the overhead of writing an entire refinement package to perform that test is overwhelming. *TNT* allows modification at a number of levels, so one can choose to work at the level that allows the easiest implementation of the idea. Several examples follow.

#### 25.2.4.6.1. *Michael Chapman's real-space refinement package*

At Florida State University, Chapman has implemented a real-space refinement package, principally intended for the refinement of virus models, using *TNT*. He was able to use *TNT*'s minimizer and stereochemical restraints unchanged along with programs he developed to implement his method. More information about this package can be found at http://www.sb.fsu.edu/∼rsref.

#### 25.2.4.6.2. *Gerard Bricogne's Buster refinement package*

Bricogne & Irwin (1996) have developed a maximum-likelihood refinement package using *TNT*. Not only are *TNT*'s minimizer and stereochemical restraints used, but many of the calculations of the maximum-likelihood residual's derivatives are performed using *TNT* programs. While Bricogne and co-workers have not needed to modify *TNT* programs to implement their ideas, there is ongoing collaboration between them and *TNT*'s authors on the development of commands that allow access to some previously internal calculations. More information about *Buster* can be found at http://lagrange.mrc-lmb.cam.ac.uk/.

#### 25.2.4.6.3. *Randy Read's maximum-likelihood function*

When Navraj Pannu wanted to implement Read's maximum-likelihood refinement functions (Pannu & Read, 1996*b*) in *TNT*, he choose not to implement it as a separate program, but modified *TNT*'s source code to create a new version of the program *Rfactor*, named *Maxfactor*.

#### 25.2.4.6.4. *J. P. Abrahams' likelihood-weighted noncrystallographic symmetry restraints*

Abrahams (1996) conceived the idea that because some amino-acid side chains can be expected to violate the noncrystallographic symmetry (NCS) of the crystal more than others, one could develop a library of the relative strength with which each atom of each residue type would be held by the NCS restraint. He chose to determine these strengths from the average of the current agreement to the NCS of all residues of the same type. For example, if the lysine side chains do not agree well with their NCS mates, the NCS will be loosely enforced for those side chains. On the other hand, if almost all the valine side chains agree well with their mates, then the NCS will be strongly enforced for the few that do not agree well.

He chose to implement this idea by modifying the source code for the *TNT* program *NCS*. Since the calculations involved in implementing this idea are simple, the extent of the modifications were not large.

## 25.2.5. The *ARP/wARP* suite for automated construction and refinement of protein models (V. S. Lamzin, A. Perrakis and K. S. Wilson)

### 25.2.5.1. *Refinement and model building are two sides of modelling a structure*

The conventional view of crystallographic refinement of macromolecules is the optimization of the parameters of a model to fit both the experimental data and a set of *a priori* stereochemical observations. The user provides the model and, although the values of its parameters are allowed to vary during the minimization cycles, the presence of the atoms is fixed, *i.e.* the addition or removal of parts of the model is not allowed. As a result, users are often faced with a situation where several atoms lie in one place, while the density maps suggest an entirely different location. Manual intervention, consisting of moving atoms to a more appropriate place using molecular graphics, density maps and geometrical assumptions can solve the problem and allow refinement to proceed further.

The *Automated Refinement Procedure* (*ARP*; Fig. 25.2.5.1) (Lamzin & Wilson, 1993, 1997; Perrakis *et al.*, 1999) challenges this classical view by addition of *real-space manipulation* of the model, mimicking user intervention *in silica.* Adding and/or deleting atoms (*model update*) and complete re-evaluation of the model to create a new one that better describes the electron density (*model reconstruction*) can achieve this aim.

#### 25.2.5.1.1. *Model update*

The quickest way to change the position of an atom substantially is not to move it, but rather involves a two-step procedure to remove it from its current (probably wrong) site and to add a new atom at a new (hopefully right) position. Such updating of the model does not imply that all rejected atoms are immediately repositioned in a new site, so the number of atoms to be added does not have to be equal to the number rejected.

*Atom rejection* in *ARP* is primarily based on the interpolated $2mF_o - \Delta F_c$ or $3F_o - 2F_c$ electron density at its atomic centre and the agreement of the atomic density distribution with a target shape.

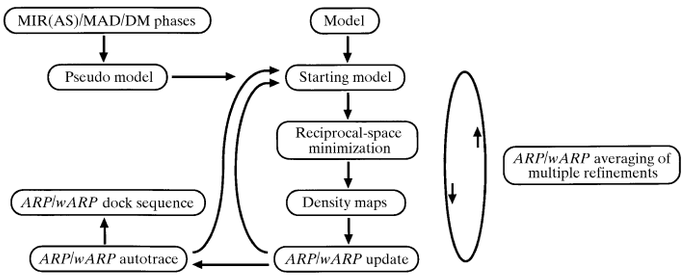


Fig. 25.2.5.1. A flow chart of the *Automated Refinement Procedure*.

Applied together, these criteria offer powerful means of identifying incorrectly placed atoms, but can suggest false positives. However, a correctly located atom that happens to be rejected should be selected again and put back in the model. Developments of further, perhaps more elegant, criteria may be expected in the future development of the technique.

*Atom addition* uses the difference $mF_o - \Delta F_c$ or $F_o - F_c$ Fourier synthesis. The selection is based on grid points rather than peaks, as the latter are often poorly defined and may overlap with neighbouring peaks or existing atoms, especially if the resolution and phases are poor. The map grid point with the highest electron density satisfying the defined distance constraints is selected as a new atom, grid points within a defined radius around this atom are rejected and the next highest grid point is selected. This is iterated until the desired number of new atoms is found and reciprocal-space minimization is used to optimize the new atomic parameters.

*Real-space refinement* based on density shape analysis around an atom can be used for the definition of the optimum atomic position. Atoms are moved to the centre of the peak using a target function that differs from that employed in reciprocal-space minimization. The function used is the sphericity of the site, which keeps an atom in the centre of the density cloud but has little influence on the *R* factor and phase quality. It is only applicable for well separated atoms and is mainly used for solvent atoms at high resolution.

*Geometrical constraints* are based on *a priori* chemical knowledge of the distances between covalently linked carbon, nitrogen and oxygen atoms (1.2 to 1.6 Å) and hydrogen-bonded atoms (2.2 to 3.3 Å). Such constraints are applied in rejection and addition of atoms.

#### 25.2.5.1.2. *Model reconstruction*

The main problem in automatically reconstructing a protein model from electron-density maps is in achieving an initial tracing of the polypeptide chain, even if the result is only partially complete. Subsequent building of side chains and filling of possible gaps is a relatively straightforward task. The complexity of the autotracing can be nicely illustrated as the well known travelling-salesman problem. Suppose one is faced with 100 trial peptide units possessing two incoming and two outgoing connections on average, which is close to what happens in a typical *ARP* refinement of a 10 kDa protein. Assuming that one of the chain ends is known and that it is possible to connect all the points regardless of the chosen route, then one is faced with the problem of choosing the best chain out of $2^{98}$. In practice, the situation is even more complex, as not all trial peptides are necessarily correctly identified in the first iteration and some may be missing – analogous to the correctness or incorrectness of the atomic positions described above.

If the connections can be assigned a probability of the peptide being correct, then only the path that visits each node exactly once and maximizes the total probability remains to be identified. Automatic density-map interpretation is based on the location of the atoms in the current model and consists of several steps. Firstly,



references