Acta Crystallographica Section D
Biological Crystallography
ISSN 0907-4449

A knowledge-driven approach for crystallographic protein model completion

Krista Joosten,[a] Serge X. Cohen,[a] Paul Emsley,[b] Wijnand Mooij,[a] Victor S. Lamzin[c] and Anastassis Perrakis[a]*

[a]Department of Molecular Carcinogenesis, Netherlands Cancer Institute, The Netherlands, [b]York Structural Biology Laboratory, Chemistry Department, The University of York, England, and [c]EMBL Hamburg Outstation, Germany

Correspondence e-mail: a.perrakis@nki.nl

Received 2 November 2007
Accepted 15 January 2008

One of the most cumbersome and time-demanding tasks in completing a protein model is building short missing regions or 'loops'. A method is presented that uses structural and electron-density information to build the most likely conformations of such loops. Using the distribution of angles and dihedral angles in pentapeptides as the driving parameters, a set of possible conformations for the C^{α} backbone of loops was generated. The most likely candidate is then selected in a hierarchical manner: new and stronger restraints are added while the loop is built. The weight of the electron-density correlation relative to geometrical considerations is gradually increased until the most likely loop is selected on map correlation alone. To conclude, the loop is refined against the electron density in real space. This is started by using structural information to trace a set of models for the C^{α} backbone of the loop. Only in later steps of the algorithm is the electron-density correlation used as a criterion to select the loop(s). Thus, this method is more robust in low-density regions than an approach using density as a primary criterion. The algorithm is implemented in a loop-building program, *Loopy*, which can be used either alone or as part of an automatic building cycle. *Loopy* can build loops of up to 14 residues in length within a couple of minutes. The average root-mean-square deviation of the C^{α} atoms in the loops built during validation was less than 0.4 Å. When implemented in the context of automated model building in *ARP*/*wARP*, *Loopy* can increase the completeness of the built models.

1. Introduction

In macromolecular X-ray crystallography, building a complete model from a density map remains a challenging task. Even though several programs exist that aim towards automated model building [for example, *ARP*/*wARP* (Perrakis *et al.*, 1999), *RESOLVE* (Terwilliger, 2003*a*), *TEXTAL* (Ioerger *et al.*, 1999), *MAID* (Levitt, 2001) and *Buccaneer* (Cowtan, 2006)], none of these programs is expected to return a 'complete' model. This means that no matter how far the automated model building has progressed, part of the model still needs to be built manually. Low-density regions, for example, can cause gaps in the model. In these regions the user has to build the model using interactive graphics, which can be quite a laborious task.

All automated model-building programs start by tracing the backbone of the protein from the density map, albeit in different manners. In *ARP*/*wARP*, for example, the electron density is first modelled by free atoms, from which the C^{α} backbone of the structure is traced (Morris *et al.*, 2002). The next step is to assign the known protein sequence to the main-chain fragments that were found ('sequence docking'; see, for example, Cohen *et al.*, 2004; Terwilliger, 2003*b*). However, at the end, even after a successful automated model-building run, some regions of known start, end, length and amino-acid sequence remain to be built.

Software that uses this information to complete the model has also been developed. For example, *Xpleo* (van den Bedem *et al.*, 2005), *LAFIRE* (Yao *et al.*, 2006) and *RAPPER* (de Bakker *et al.*, 2006) are tools for automated model completion. In this paper, we describe the use of structural information on the C^α backbone to fill in the gaps and build the loops of the model. It has been shown (Jones *et al.*, 1991) that a database of five-residue-long fragments (pentapeptides) could be used to complete and improve the backbone structure found by skeletonization of the density map. It has also been shown that protein conformation can be described by the angles and dihedral angles between successive C^α atoms (Kleywegt, 1997; Esnouf, 1997).

We use knowledge from pentapeptides to predict the probable positions of the fifth C^α atom from the terminal tetrapeptides of main-chain fragments, thus extending the peptide segment. By iterating this process, using each set of new C^α atoms as a new set of terminal tetrapeptides, we create a tree of possible backbones for the loop. In several steps based on different features we remove less likely options until the most probable loop(s) is/are selected from the tree. Our loop-building method can be used in two modes. In the first case, gaps are automatically detected, the best loop for each gap is selected and the resulting model is returned to the user. In this mode, *Loopy* can easily be incorporated into an automated package. In the second case, the user can define an area to (re-)build. The program now provides a selection of the best possible loops ordered by density correlation. This allows the creation of an ensemble of models (Terwilliger *et al.*, 2007; de Bakker *et al.*, 2006; DePristo *et al.*, 2004; Furnham *et al.*, 2006), at least locally; this selection can also aid the user in building the loop manually by interactively inspecting and editing the given choices.

Since our approach is based on the usage of structural information from pentapeptides, we will start with a description of the function we use and how we obtained our data. Next, we will explain the method we use to build the loops. Results of testing *Loopy* on manual builds as well as as a part of *ARP*/*wARP* are then described. Finally, we summarize our results on *Loopy* and add a few ideas for further improvements.

2. Structure of a five-C^α fragment (pentapetide)

Consider a missing region in the middle of a protein model (for simplicity, we will refer to such regions as 'loops'). When all the fragments of the model are docked into sequence, both anchors of the loop (the preceding and succeeding residues) are known and the number of missing residues is also known. Moreover, not only the position and the amino-acid type of the anchors are known, but the geometrical features of the protein model close to the anchors are also known. Our aim is to use these features to extend the anchors over the number of missing residues, effectively filling in the gap.

As the main geometrical feature, we consider a fragment of four C^α atoms (or tetrapeptide; see Fig. 1). We define a tetrapeptide from the N-terminus to the C-terminus as 'forward' and one in the opposite direction as 'backward'. Initially, we investigated whether one can predict the position of the fourth C^α atom given the positions of the first three C^α atoms of a tetrapeptide. This tripeptide defines an origin and a natural basis. Under the assumption that the variation in the distance between successive C^α atoms, d (3.8 Å), is negligible, the relative position of the fourth C^α atom can be described by (d, φ_0, θ_0, φ_1). Likewise, the relative position of the fourth C^α atom in a backward tetrapeptide is given by (d, φ_1, θ_0, φ_0).

The density profiles of the angle and torsion (φ_1, θ_0) for forward tetrapeptides and (φ_0, θ_0) for backward tetrapeptides have been studied in the PhD thesis of R. Morris (Morris, 2000); his results are illustrated in Fig. 2 and clearly show two separate peaks: a sharp peak representing the α-helices (at θ_0 around 50°) and a broad peak representing β-strands (at θ_0 around −150°). The first peak is roughly a factor of one hundred higher than the second peak. This suggests that trying to predict the position of the fourth C^α atom using these plots would strongly favour α-helices, even when we include φ_0 for forward and φ_1 for backward peptides. We checked this suggestion and concluded that we indeed needed to use larger structural fragments to provide additional information.

Therefore, we decided to consider five-C^α fragments (pentapeptides) instead of tetrapeptides. The fifth C^α atom of a forward pentapeptide can be described in terms of (d, φ_0, θ_0, φ_1, θ_1, φ_2) or similarly for a backward pentapeptide in terms of (d, φ_2, θ_1, φ_1, θ_0, φ_0) (see Fig. 1).

To determine the frequency tables for the given combinations of angles and torsions, we downloaded all structures present in the PDB on 12 October 2005. From these, we kept for our learning set all structures with a reported $R_{\rm work}$ better than 25% that had been refined at a resolution higher than 2.0 Å, leaving a set of approximately 12 000 structures. These structures were then randomly distributed over ten sets each containing 1200 structures. For every protein, we computed the angles and dihedral angles for every possible pentapeptide in both the forward and the backward directions. The results of these analyses were tabulated in multi-dimensional tables; angles (φ) between 75 and 155° were binned every 10° and dihedral angles (θ) between −180 and 180° every were binned 15°. The variation between the frequency tables derived from

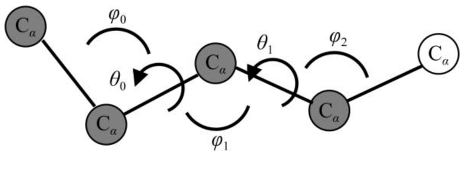

Figure 1
Angles and dihedral angles of the C^α atoms in a tetrapeptide or pentapeptide.

each of the ten sets was used to estimate the accuracy of the average of all values and look-up tables were constructed. We found that the variation in φ_0 in the forward direction and in φ_2 in the backward direction was negligible. Hence, we could approximate the propensity distributions to the four-dimensional tables of (θ_0, φ_1, θ_1, φ_2) in the forward direction and (θ_1, φ_1, θ_0, φ_0) in the backward direction.

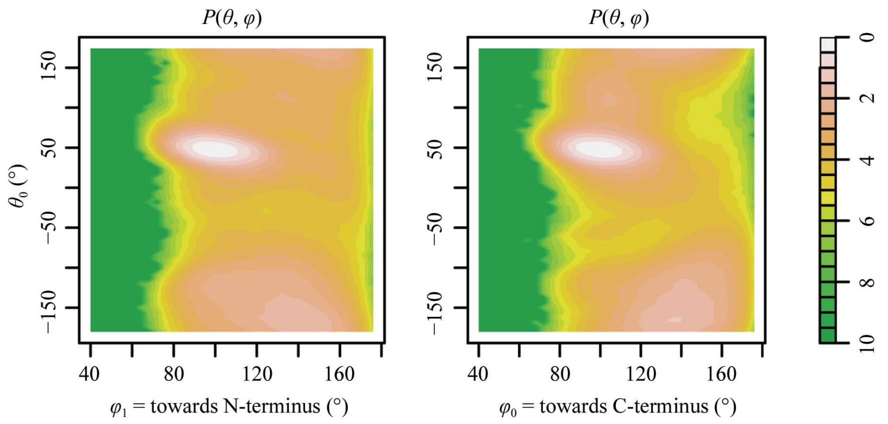

Figure 2
Logarithmic occurrence of angle and dihedral angle of the C$^\alpha$ atoms in tetrapeptides.

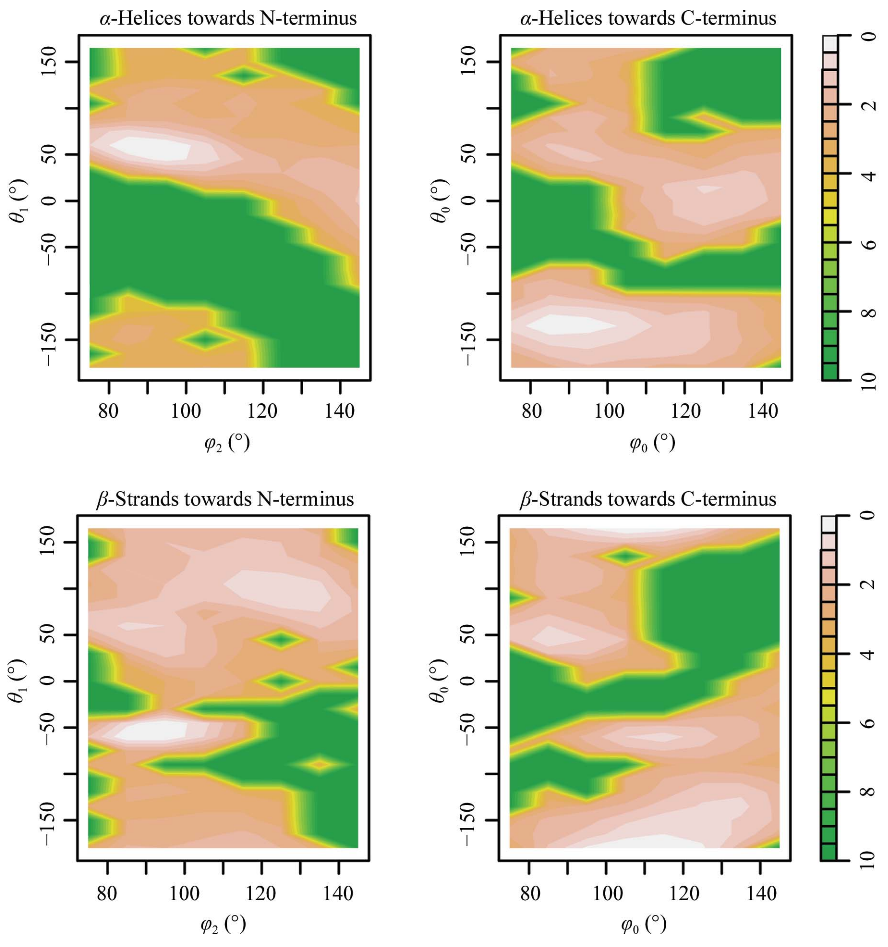

Figure 3
Examples of the logarithmic occurrence of angle and dihedral angle of the C$^\alpha$ atoms in pentapeptides.

To ensure that these tables reflect changes in the secondary structure, we displayed (as in Fig. 3) the retrieved density profiles in the regions of α-helices and β-strands. The panels show clearly that the density profile depends strongly on the secondary structure at a given position in the protein.

How can we use these density tables to predict the position of a fifth C$^\alpha$ atom given a tetrapeptide? Let us consider the C-terminus of a fragment in a protein model. The final tetrapeptide at this end of the fragment can be seen as the start of a forward pentapeptide. The C$^\alpha$-atom positions of this part of the pentapeptide are known and thus (θ_0, φ_1) are fixed. From Fig. 3 we know that the θ_0 torsion and the φ_1 angle describe the conformation of this part of the fragment effectively. The probability that the fifth C$^\alpha$ atom lies at an angle φ_2 and a dihedral angle θ_1 to the tetrapeptide is given by

$$P(\varphi_2, \theta_1 | \varphi_1, \theta_0) = \frac{P(\varphi_1, \varphi_2, \theta_0, \theta_1)}{P(\varphi_1, \theta_0)} \quad (1)$$

when we use our observation that φ_0 is independent of the other angles and torsions.

Equivalently, the final tetrapeptide at the N-terminus of the fragment can be seen as the start of a backward pentapeptide. In this frame of reference, we found φ_2 to be independent of the other angles and torsions. Thus, we find for the probability that (φ_0, θ_1) describes the position of the fifth C$^\alpha$ atom

$$P(\varphi_0, \theta_0 | \varphi_1, \theta_1) = \frac{P(\varphi_0, \varphi_1, \theta_0, \theta_1)}{P(\varphi_1, \theta_1)}. \quad (2)$$

In the next section, we explain how we use the structural information to extend a fragment by a single C$^\alpha$ atom and iterate it to obtain loops.

3. Method

Let us consider two main-chain fragments which are docked into sequence. The gap between these successive fragments is n residues long. Let us call the C$^\alpha$ atom of the N-terminus of the loop the 'N-anchor' of the loop to be built. It is connected to the C-terminus of the preceding main-chain fragment. By extending this fragment iteratively in the forward direction, a connection can be made with the succeeding fragment. We call the C$^\alpha$ atom of the second anchor point the 'C-anchor' of the loop.

Of course, a loop can also be built in the opposite direction by iterating the extension from the C-anchor backwards.

In our loop-building algorithm, the creation of C^α backbones for possible loops is initially geometry-driven. Initially, the electron-density map plays the role of a 'mask'; it is used to avoid building over the existing model by applying negative density around the atoms of the model. As the 'tree' of possible conformations is reduced to the most likely candidates in a hierarchical manner, the contribution of the electron-density correlation to the selection criteria is slowly increased. The value of the correlation is only used to compare loops with each other; it is not used as a global measure. As a result, this method can easily bridge areas of low electron density using geometry, while at the end loops are finally selected based on the electron-density map.

A flowchart of our algorithm is given in Fig. 4. A short description of the algorithm steps is given below; each step is discussed separately in the following sections.

(i) Select a small number of possible C^α-atom positions, p, based on the structural information; these are likely extensions of a fragment by a single C^α atom. This step is iterated n times, creating a tree of possible loop backbones.

(ii) The large tree (which initially contains approximately p^n conformations or 'paths') is pruned by removing the most unlikely 'branches'. Removing a single branch, which can be furcated, can result in the removal of multiple paths. Loops leading far away from the opposite anchor are removed, as well as loops with a geometrically unfavourable connection with the opposite anchor and those with a relatively low C^α electron-density correlation.

(iii) Determine the position of all main-chain atoms of the remaining loop conformations. In this step, we remove paths that are unlikely based on failure to find a peptide plane for the main-chain atoms, unlikely Ramachandran dihedral angles or too low density correlation of all main-chain atoms. Steps (i)–(iii) are performed twice, once for each direction in which the loop can be built (forward and backward). This is important as the tree of possible branches depends strongly on the structure of the anchor tetrapeptides.

(iv) Build the side chains of all peptides in the remaining loops. The best loop(s) is/are selected based on the density correlation of the all the atoms in the loop.

(v) Finally, the best loop(s) are refined in real space.

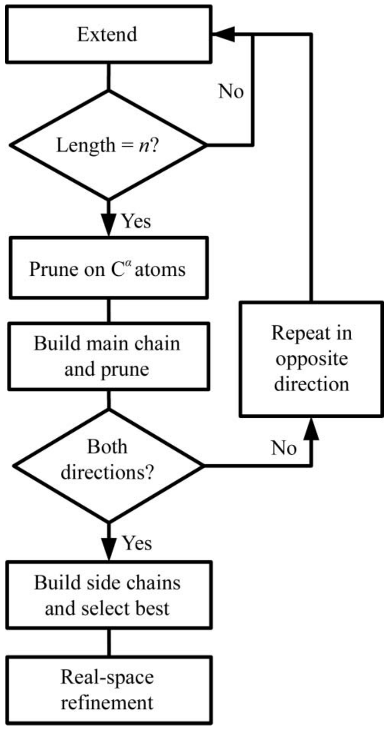
Extend
Length = n?
No
Yes
Prune on C^α atoms
Build main chain and prune
Repeat in opposite direction
Both directions?
No
Yes
Build side chains and select best
Real-space refinement

Figure 4
Flowchart of *Loopy*.

3.1. Extending fragments by a single C^α atom at a time

We start by creating a 'real-space' residual map, as shown in Jones & Liljas (1984) and described in a similar implementation in Cohen *et al.* (2004). This sets the map density around each atom of the existing model to a negative value. Next, we create a uniform spherical grid with radius d (3.8 Å) around one of the anchor points of our loop. The algorithm we use to create a uniform spherical grid restricts the number of grid points to the Fibonacci sequence. Tests showed that the performance was optimal for 377 nodes. For each node in the grid, we determine the density at the node itself and midway between the node and the anchor. Since we expect at least some density at both points, we remove nodes with negative density at either position (the density map has the mean set to zero, as commonly performed). We have found that including the requirement for positive density at the midway point strongly improved the performance of the program. Note also that this is a very generous density constraint, which mostly

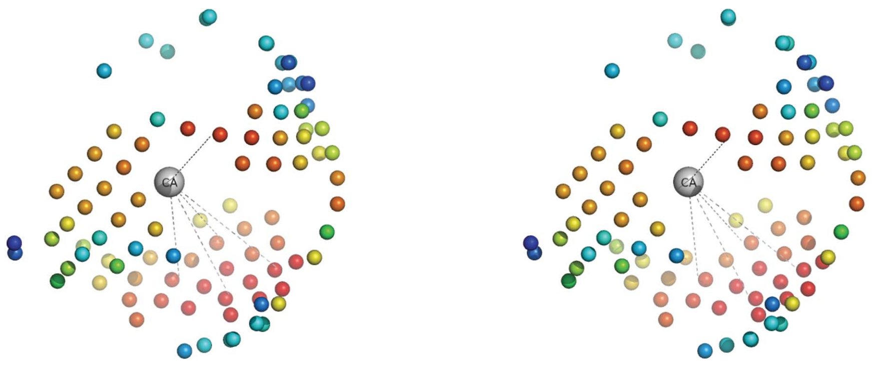

Figure 5
Example of possible points p from an existing candidate terminal C^α atom (large grey sphere) in stereographic view. Candidate positions are shown as smaller spheres using a 'heat' colour scheme: blue refers to a low score and red to a high score.

ensures that we do not build over existing fragments or over the fragment itself.

For the remaining nodes, we compute the angle and torsion (φ, θ) and look up the corresponding structural probability from our tables. We use this value in combination with the electron density at the node to score the nodes. The contribution of the electron density is approximately a factor of ten smaller than the structural probability (the relative weights are set empirically) and thus the contribution of the density only involves the fine-tuning of the scoring.

Since the above procedure is iterated n times, the number of nodes may lead to a combinatorial explosion, p^n. Thus, it is desirable to keep the number of nodes to a minimum. However, we cannot simply select nodes with the highest score. Consider the example displayed in Fig. 5: the figure shows that nodes tend to cluster together, with a single cluster having several high-scoring nodes (red spheres; lower scores are blue). We try to select the representative nodes from each cluster by taking the one with the highest score within a sphere of 0.3 Å radius. We found that in general five nodes per extension sphere suffice to build most loops.

3.2. Removing unlikely conformations

The single extension for the p node candidates n times creates a tree of possibly up to p^n C$^\alpha$ atoms representing possible backbones for the loop. The tree can grow in every direction, restricted only by the structural probability and positive electron density. In this step, we prune the tree by removing the least likely paths as follows.

(i) Those ending too far away from the opposite loop anchor are removed.

(ii) Loop paths are ordered according to the density at each C$^\alpha$ node and their density midway from the edges. Only a best selection is kept (typically around 100).

(iii) Structural probability is determined for the connection of each loop end node to the opposite anchor point. Again, the branches are ordered according to the structural probability and the best ones are kept (typically a few dozen).

3.3. Building the main-chain atoms

After the first round of pruning, we determine the position of the main-chain atoms for each residue in the loop tree. The main-chain atoms of a peptide lie approximately in a plane. Their relative positions are known to high accuracy and with negligible variations within this plane. Currently, we only consider *trans*-peptides.

For all amino acids except glycine, the position of the anchor C$^\beta$ atom can be determined from the positions of the C and the N atom around the anchor C$^\alpha$ atom. Furthermore, little or no density is expected outside the peptide plane. We selected four points ($E^+_{\rm max}$, $E^-_{\rm max}$, $E^+_{\rm min}$, $E^-_{\rm min}$) which we found to have low density in comparison to the main-chain atom positions. The exact location of these points was chosen such that the procedure gave optimal results for the reproduction of the orientation for each peptide plane in the structure 1lml at a resolution of 2.0 Å. This yields eight (or seven for glycine) points to determine the orientation of the peptide plane between two successive C$^\alpha$ atoms.

Let $\rho(x)$ be the density at the position of atom x. The plane is rotated through two successive nodes, maximizing the value of

$$\begin{aligned}\rho_{\rm total} = {}& \rho({\rm C}) + \rho({\rm O}) + \rho({\rm N}) + \rho({\rm C}^\beta)\\ & - \rho(E^+_{\rm min}) - \rho(E^-_{\rm min}) - \rho(E^+_{\rm max}) - \rho(E^-_{\rm max}). \quad (3)\end{aligned}$$

This search is restricted by the constraint that the angle between N, C$^\alpha$ and C should be $109 \pm 20^\circ$. If no peptide plane is found which complies with this restraint, it is assumed that the corresponding C$^\alpha$ candidates are wrong and all paths containing this edge in the tree are removed.

We complete the loop tree with the main-chain atoms and remove those branches that are unlikely in the following steps.

(i) Determine the most probable position of the main-chain atoms by rotating the peptide plane as described above. Branches with residues for which we cannot find a plane are completely removed from the tree.

(ii) Determine the Ramachandran angles (Ramachandran *et al.*, 1963) for the residues in every possible loop. In this reduction, glycines aside, loops which include peptides with a value of zero in the four-valued Ramachandran plot described in Kelly (2008) are removed.

(iii) Order the loops based on the density correlation of all main-chain atoms and, if present, the C$^\beta$ atoms. Typically, about five loops are kept for the final selection step.

3.4. Completion of the loop

After the latter selection step, in which we were still working both in the forward and the backward directions, only a small number of possible loop paths remain. For this final selection, the positions of all side-chain atoms in each candidate loop are determined using the algorithm from *ARP/wARP*. A weighted combination of the density correlation of the side-chain atoms and that of the main-chain atoms is used for the final scoring of the paths.

3.5. Real-space refinement

After a loop with all side chains has been built, we refine it in real space. Real-space refinement for the purpose of this study has been performed as implemented in the program *Coot* (Emsley & Cowtan, 2004) utilizing a script that refines the loop in real space with geometrical restraints.

3.6. Implementation and hardware details

Loopy was written in C++. The ATLAS library (Whaley & Petitet, 2005) provided us with an automatically optimized *BLAS/LAPACK* implementation for linear algebraic computations. *Clipper* (Cowtan, 2003) was used for handling electron-density maps. In the initial stage of the algorithm a linear interpolation was used to determine the electron-density correlation. This method is ten times faster than the

Gaussian interpolation method that we use in the final stage, although less accurate. However, at the beginning many map correlations need to be determined and accuracy is less important at this stage than speed. In the final stage, where we determine the side-chain atoms and score all loops, we need a better and more precise interpolation method. In this stage, we compute the electron-density correlation by approximating the atoms by Gaussian distributions centred at the atomic position (Cowtan, 2003).

Loopy compiles under different architectures and operating systems, including Linux, Mac OSC, Alpha Tru64 Unix and SGI Irix. The validation was performed on an Intel Xeon 2.66 GHz machine under Fedora core 5. Testing in the context of *ARP*/*wARP* model building was performed on an Intel Pentium 4 3.00 GHz and on a cluster of five Apple X-serve G5 nodes.

Table 1
Nodes per extension and average time per number of residues.

The default choices for the maximum number of point per extension, $p_{\max}$, for each loop length are shown in bold. Time is the average time per loop for about 100 loops in each case, including real-space refinement. The success rate is calculated as the percentage of times that *Loopy* actually returned a solution. R.m.s.d. C^α is the root-mean-square deviation from the final structure for all built loops after real-space refinement.

		$p_{\max}$				
Length		2	3	4	5	6
4	Time (h:min:s)	0:0:10	0:0:12	0:0:14	**0:0:15**	0:0:16
	Success (%)	98.1	99.1	99.1	**99.1**	99.1
	R.m.s.d. C^α (Å)	0.08	0.06	0.06	**0.05**	0.06
6	Time (h:min:s)	0:0:13	0:0:19	**0:0:30**	0:0:52	0:1:24
	Success (%)	94.3	99.1	**99.1**	100.0	100.0
	R.m.s.d. C^α (Å)	0.12	0.09	**0.08**	0.08	0.07
8	Time (h:min:s)	0:0:17	0:0:46	**0:2:49**		
	Success (%)	93.4	100.0	**100.0**		
	R.m.s.d. C^α (Å)	0.29	0.08	**0.06**		
10	Time (h:min:s)	0:0:25	**0:3:20**			
	Success (%)	85.7	**99.0**			
	R.m.s.d. C^α (Å)	0.25	**0.08**			
12	Time(h:min:s)	0:0:50	**0:6:01**			
	Success(%)	76.9	**97.1**			
	R.m.s.d. C^α (Å)	0.37	**0.09**			
14	Time (h:min:s)	**0:2:14**	2–3 h			
	Success (%)	**76.7**	—			
	R.m.s.d. C^α (Å)	**0.37**	—			

4. Results

The results of our program were examined in two stages. Firstly, we wanted to validate *Loopy*: could the program rebuild parts of a model using the best available map? In the next stage, we considered a more interesting question: can *Loopy* build loops in difficult parts of the density map? In other words, could *Loopy* add to the model completion by building loops where another program, in this case *ARP*/*wARP*, failed? All these tests will be described in the next subsections.

4.1. Validation

We validated *Loopy* on two structures: 1lml and 1o1z. Both structures had been refined to a resolution of 2.0 Å. For the initial validation of *Loopy*, we ran it in the manual mode to rebuild a few random parts of 1lml and checked the generated loop suggestions visually. In Fig. 6, a representative example of a loop test is shown for a loop with anchor points at residues 35 and 43. In this example, all loop suggestions closely resemble the final model (pink). The loop with the highest score (green) is very close to the reference structure.

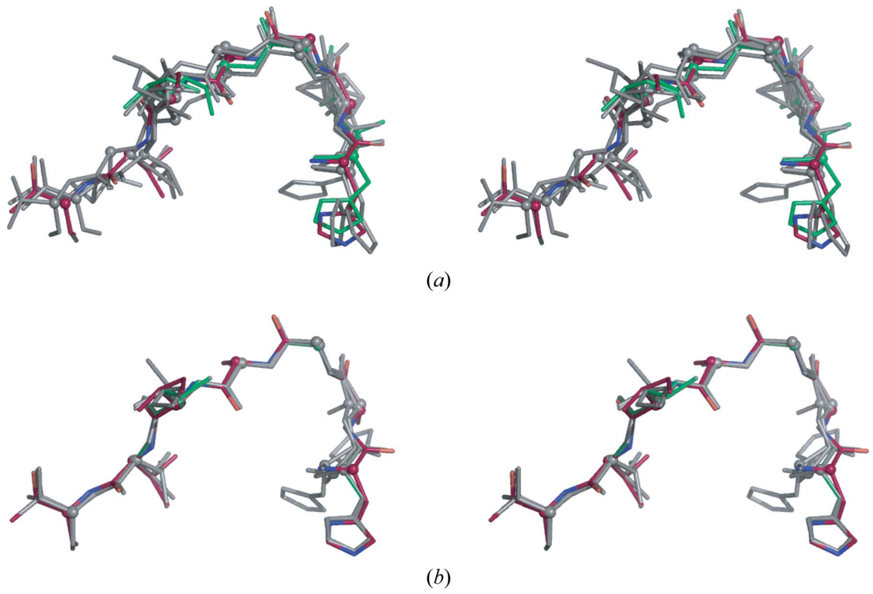

Figure 6
Example of loops for 1lml between residue 35 and 43 in stereographic view. The pink loop is the reference structure, the green loop is the loop with the highest score and grey loops are alternative loops. (*a*) Before and (*b*) after real-space refinement.

For a more extensive validation, we used the structure 1o1z which had also been used for the validation of *Xpleo* (van den Bedem *et al.*, 2005). We rebuilt every polypeptide in the final structure of 4, 6, 8, 10, 12 and 14 residues long. Results for the various loop lengths are given in Fig. 7, which displays a box plot for the r.m.s.d. of all atoms after real-space refinement. Each time around 100 loops were built. In Table 1 we show how various settings of the maximum number of nodes per extension and loop length affect the average building time, success rate and accuracy of the loops. Based on these tests, we adjusted the default number of maximum number of nodes per extension to balance time, success and accuracy (defaults are highlighted in Table 1).

We also showed that the r.m.s.d. of the C^α atoms alone gives a good indication of the quality of the loop. Furthermore, real-space

Table 2
The results of the testing protocol described in §4.2.

R, resolution; *f*, number of residues in the final model. We firstly show the results after running the main-chain and side-chain building modules of *ARP*/*wARP* alone (*C*, chains; *r*, correct residues) and then we indicate the same numbers after running *Loopy* [*C*(*L*), *r*(*L*)]; *t*, time.

Protein	*R* (Å)	*f*	*C*	*r*	*C*(*L*)	*r*(*L*)	*t* (s)
2abb	1.0	361	7	334	1	356	51
2jae	1.3	955	17	909	6	929	18
1i12	1.3	619	8	579	6	585	16
1wpn	1.3	374	16	312	8	342	84
1zua	1.3	317	11	260	1	307	30
2*a*50	1.3	457	21	386	8	425	131
2nw8	1.6	534	8	495	7	497	141
1fgo	1.6	817	22	731	3	791	53
1h5a	1.6	306	13	259	1	303	29
1ou8	1.6	231	5	215	4	217	14
1tm7	1.6	345	4	303	2	333	102
2b3k	1.6	304	5	287	2	294	94
2b9h	1.6	349	5	329	3	334	66
1j4a	1.9	1325	19	1171	12	1188	72
2fsa	1.9	507	6	481	3	492	37
2ij3	1.9	907	17	842	5	885	20
2aka	1.9	1069	17	931	7	966	74
1oim	2.2	887	12	852	3	868	40
2*aa*5	2.2	510	7	485	4	491	309
1zrq	2.2	836	17	753	12	767	88
1vg0	2.2	663	9	640	5	650	126
1e8h	2.6	1090	25	1002	10	1029	42
1o70	2.6	296	5	279	1	292	81
2arh	2.6	584	16	503	9	520	13
2bvm	2.6	541	8	504	2	520	50
1gmo	3.0	1369	73	820	67	828	242
2bxr	3.0	890	40	590	21	604	96
1b9x	3.0	577	16	444	8	460	28
1r5o	3.0	409	16	258	6	273	31
1yhn	3.0	248	7	186	4	193	7
1zy1	3.0	388	18	292	8	321	16
2a2z	3.0	885	31	742	21	754	235
2deo	3.0	399	12	310	10	313	63
1s78	3.3	1995	72	952	39	974	232
1j1e	3.3	720	14	164	14	164	99
2dcu	3.3	544	24	293	13	312	11
2ffl	3.3	2896	105	1839	76	1844	126

refinement improves the loops considerably, demonstrating that despite the inherent inaccuracy of its building algorithm, *Loopy* places the loops accurately enough to be positioned correctly by real-space refinement. Details of this are illustrated in Fig. 1 of the supplementary material[1].

The validation gave us confidence to run the test as part of an automatic building procedure.

4.2. Loop building after *ARP*/*wARP* main-chain tracing

To test how *Loopy* performs in the most difficult regions of a structure, which are typically those that need to be built in order to obtain a complete model, we devised the following protocol. The main-chain tracing and side-chain tracing modules of *ARP*/*wARP* were run on the best available map using as free atoms the atomic coordinates from the final structure, only once and without any iteration with refinement.

[1] Supplementary material has been deposited in the IUCr electronic archive (Reference: GX5125). Services for accessing this material are described at the back of the journal.

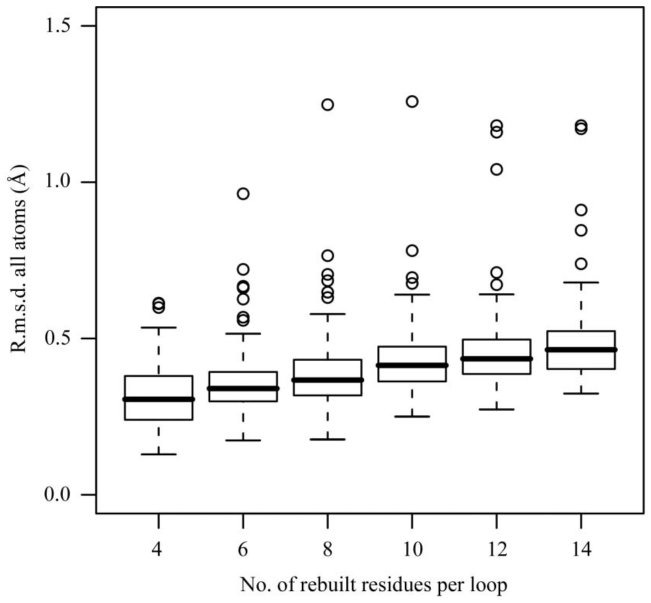

Figure 7
For validation purposes, portions of the original structure were rebuilt. This plot shows the r.m.s.d. of all atoms in a loop *versus* its number of residues. The median is displayed as a thick bar, the first and third quartile are represented by the rectangle and the minimum and maximum are shown by the caps of the dotted lines. Outliers are determined as 1.5 times the interquartile range. They are displayed as circles.

This is the best possible scenario for the *ARP*/*wARP* tracing modules: the best map and the most accurate free-atom coordinates are used. Under these conditions, the regions of the structure that are not built by the *ARP*/*wARP* main tracing modules would almost certainly never be built by *ARP*/*wARP* and thus represent the most challenging areas of the structure. The test set was composed of a broad range of different structures for which we know the final model. The set contained 38 structures in total and included structures with multiple chains and/or NCS. The resolution of the data sets ranged from 1.0 to 3.3 Å.

In the automatic mode *Loopy* determines the loop positions from the PDB file of the model; the sequence of the structure is provided in a PIR format file, like in a classic *ARP*/*wARP* run. *Loopy* then tries to build all loops according to the user input. Before starting to build each loop, the program checks that the anchors or their symmetry mates lie within reasonable distance. For our test, we used a maximum loop length of 14 residues. The value of *p* (see §3.1) was varied with the length of the loop, starting between six for short loops and two for the longest loops. The loop quality was determined as the r.m.s.d. between the C^α atoms in the loops and those in the reference structure. Since we expect that the anchors are wrong when the gap consists of only one or two residues, for these small loops the gap was broadened on each side if possible. This effectively meant that we rebuilt the initial anchors.

Table 2 shows that *Loopy* can build difficult regions of the structure. Thus, it adds many additional correct C^α atoms to the structure built by the protocol described in §4.2 without adding too many incorrect atoms. It should also be noted that in an iterative *ARP*/*wARP* run the wrong loops would be most likely removed or, in the context of *flex-wARP*, flagged as wrong by *ElAl* (Cohen *et al.*, 2004). The time needed per structure varied from a couple of seconds to 4 min at the most.

To quantify the quality of the built loops, we studied the r.m.s.d. of the C^α atoms per loop (see Fig. 8) and observed a decrease in the accuracy of the loops with resolution. For a resolution <2.0 Å the median of the r.m.s.d. lies at 0.2 Å and it increases to 0.8 Å for resolutions lower than 3 Å. We furthermore observe that the value of the r.m.s.d. increases with the length of the loop; since only a few loops with a length of eight residues or more are available, the decrease in the median r.m.s.d. is hardly significant. In supplementary Fig. 2, we show that the percentage of C^α atoms in each loop that lie within 0.7 Å of the reference structure is also very high.

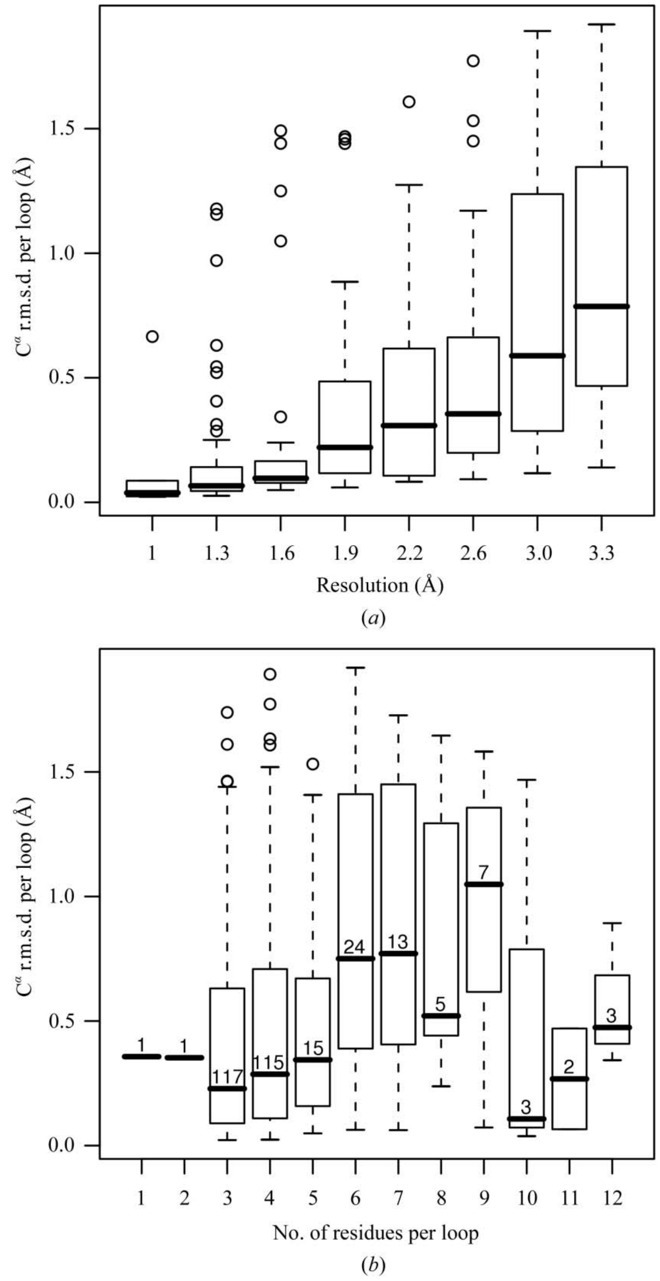

Figure 8
The C^α r.m.s.d. of the loops built by *Loopy* on models computed using the protocol described in §4.2 are displayed against the resolution (*a*) and the loop length (*b*). The C^α r.m.s.d. clearly increases with the resolution. For the loop length, the number of loops per length (given above the median line in the box plot) is insufficient to derive any statistical conclusions for the C^α r.m.s.d. as a function of the loop length. (See the caption of Fig. 7 for an explanation of the box plot).

4.3. Loop building within the *ARP/wARP flex-wARP* module

As a final test, we implemented *Loopy* within the *ARP/wARP flex-wARP* module (Cohen *et al.*, 2004) that was introduced in *ARP/wARP* release 7.0 (July 2007). In this flexible module of *ARP/wARP*, we could not only implement *Loopy* after the full *ARP/wARP* run but could also run *Loopy* internally after each model-building cycle. In this test, we want to establish whether the overall completeness of the models delivered by *ARP/wARP* increases when *Loopy* is run in every *ARP/wARP* model-building cycle. More precisely, each time the *ARP/wARP* main-chain tracing module is run and the resulting main-chain fragments are docked into sequence, all possible loops shorter than ten residues are built. After the final main-chain tracing cycle all possible loops shorter than 14 residues are built. For a set of 30 structures from the *ARP/wARP* test-cases deposition site (http://xtal.nki.nl/Depot), the best possible results were compared with those of a *flex-wARP* run without *Loopy*. The results are displayed in Fig. 9. Usage of *Loopy* generally increases the completeness of the models, sometimes significantly, especially for higher resolution. In a small number of the cases the number of residues found using *Loopy* was slightly reduced.

5. Discussion

We have shown that using the geometry of pentapeptides as a driving force combined with a hierarchical pruning algorithm is a powerful and accurate method for loop building. The validation of *Loopy* showed that for high-resolution data *Loopy* can rebuild parts of a structure up to 14 residues long with a median accuracy below 0.5 Å within a couple of minutes on a 2.66 GHz Pentium 4.

During the tests on 'real' loops, we found that *Loopy* can build difficult regions and add to model completeness, espe-

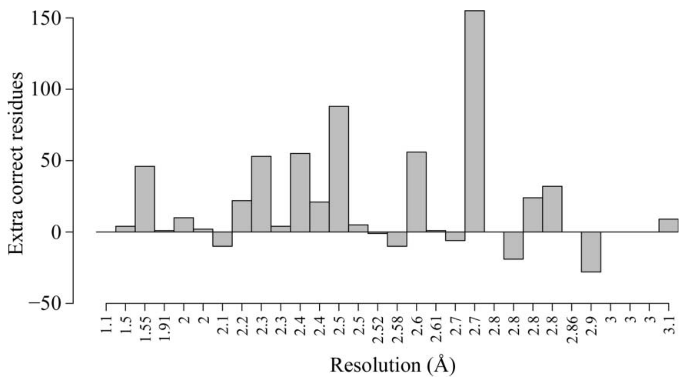

Figure 9
Extra correct residues built with *Loopy* integrated within *flex-wARP* compared with models created by *flex-wARP* without *Loopy*.

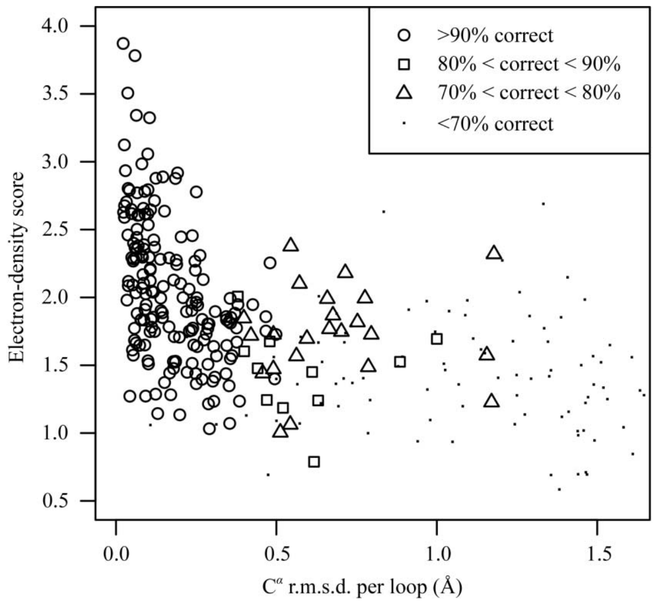

Figure 10
The scoring function is plotted against the r.m.s.d. of the C$^\alpha$ atoms per loop built by *Loopy* after the protocol described in §4.2. The figure displays a negative correlation. The symbols in the plot represent the percentage of C$^\alpha$ atoms in a loop that are within 0.7 Å of the reference.

cially at resolutions higher than 2.6 Å. The average r.m.s.d. of the C$^\alpha$ atoms in the loops lies around 0.5 Å, although the r.m.s.d. increases with the loop length. For lower resolution data, *Loopy* still improves the model completeness, but the number of C$^\alpha$ atoms further than 0.7 Å from their equivalents in the reference structure increases. Note, however, that at lower resolution the accuracy of position of the atoms in the reference structure also decreases.

In the future, we would like to improve the performance of *Loopy*, in particular for lower resolution and longer loops. We have found that in these cases a failure to find the correct loop often originates from the pruning algorithm. When we verified our scoring function (see Fig. 10), we found that its (negative) correlation with the r.m.s.d. is not very strong. As a result, we are at present not able to use the scoring as a measure of accuracy or as an indicator for incorrect loops. We aim to rectify this in the future. Furthermore, we have found that for lower resolutions and for longer loops our method for finding main-chain atoms often fails to find any possible peptide plane. This may be caused by the selection criteria used in previous steps of the pruning algorithm or by the plane-search algorithm. We plan to study this problem and to try to improve the robustness of this part of the algorithm. Finally, we would like to remedy the exponential time and memory usage of *Loopy*. One idea is to check the distance to the opposite anchor point during the building of the tree and remove the suggested C$^\alpha$ atoms that are unlikely based on that distance. This should reduce the size of the loop tree generated and thereby reduce both the time and the memory usage. A first test with this idea shows that it will indeed increase the speed and as a side effect improve the accuracy of the loops. As a final remark we would like to note that *Loopy* is currently unable to build *cis*-peptides. The method we use to search the plane of the main-chain atoms of the peptide implies *trans*-peptides. We expect as well that the angles and torsions of a pentapeptide that includes a *cis*-peptide will deviate from the data that we have acquired.

Overall, we conclude that *Loopy* is useful software that increases model completeness in automated model building. It is also a valuable tool for suggesting loop conformations during manual model building. Finally, it can facilitate the building of ensembles of loops and testing the idea of refining partial model ensembles along the lines suggested by Furnham *et al.* (2006).

We would like to thank Richard J. Morris for providing the tables for the tetrapeptides. This research was supported by the NIH (grant No. R01 GM62612). SXC thanks NWO/CW (VENI 700.55.405). This work was performed under the coordination of the EU BIOXHIT FW6 Integrated Project (grant No. LSHG-CT-2003-503420).

References

Bakker, P. I. de, Furnham, N., Blundell, T. L. & DePristo, M. A. (2006). *Curr. Opin. Struct. Biol.* **16**, 160–165.

Bedem, H. van den, Lotan, I., Latombe, J.-C. & Deacon, A. M. (2005). *Acta Cryst.* D**61**, 2–13.

Cohen, S. X., Morris, R. J., Fernandez, F. J., Ben Jelloul, M., Kakaris, M., Parthasarathy, V., Lamzin, V. S., Kleywegt, G. J. & Perrakis, A. (2004). *Acta Cryst.* D**60**, 2222–2229.

Cowtan, K. (2003). *IUCr CompComm. Newslett.* **2**, 4–9.

Cowtan, K. (2006). *Acta Cryst.* D**62**, 1002–1011.

DePristo, M. A., de Bakker, P. I. & Blundell, T. L. (2004). *Structure*, **12**, 831–838.

Emsley, P. & Cowtan, K. (2004). *Acta Cryst.* D**60**, 2126–2132.

Esnouf, R. M. (1997). *Acta Cryst.* D**53**, 665–672.

Furnham, N., Blundell, T. L., DePristo, M. A. & Terwilliger, T. C. (2006). *Nature Struct. Mol. Biol.* **13**, 184–185.

Ioerger, T. R., Holton, T., Christopher, J. A. & Sacchettini, J. C. (1999). *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, edited by T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H.-W. Mewes & R. Zimmer, pp. 130–137. Menlo Park, USA: AAAI Press.

Jones, T. A. & Liljas, L. (1984). *Acta Cryst.* A**40**, 50–57.

Jones, T. A., Zou, J.-Y., Cowan, S. W. & Kjeldgaard, M. (1991). *Acta Cryst.* A**47**, 110–119.

Kelly, K. (2008). *Protein Structure Validation and Analysis.* http://www.chemcomp.com/journal/provalid.htm.

Kleywegt, G. J. (1997). *J. Mol. Biol.* **273**, 371–376.

Levitt, D. G. (2001). *Acta Cryst.* D**57**, 1013–1019.

Morris, R. J. (2000). PhD thesis. Karl-Franzens-Universität, Graz, Austria.

Morris, R. J., Perrakis, A. & Lamzin, V. S. (2002). *Acta Cryst.* D**58**, 968–975.

Perrakis, A., Morris, R. & Lamzin, V. S. (1999). *Nature Struct. Biol.* **6**, 458–463.

Ramachandran, G. N., Ramakrishnan, C. & Sasisekharan, V. (1963). *J. Mol. Biol.* **7**, 95–99.

Terwilliger, T. C. (2003*a*). *Acta Cryst.* D**59**, 38–44.

Terwilliger, T. C. (2003*b*). *Acta Cryst.* D**59**, 45–49.

Terwilliger, T. C., Grosse-Kunstleve, R. W., Afonine, P. V., Adams, P. D., Moriarty, N. W., Zwart, P., Read, R. J., Turk, D. & Hung, L.-W. (2007). *Acta Cryst.* D**63**, 597–610.

Whaley, R. C. & Petitet, A. (2005). *Software Pract. Exp.* **35**, 101–121.

Yao, M., Zhou, Y. & Tanaka, I. (2006). *Acta Cryst.* D**62**, 189–196.