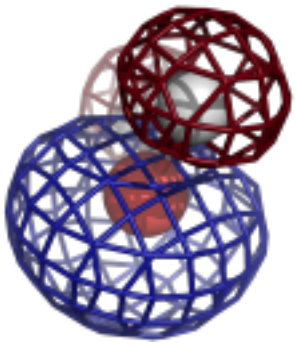
VOLUME SEVEN



COMPUTATIONAL CRYSTALLOGRAPHY NEWSLETTER

JANUARY MMXVI

HOOGSTEEN, BASE PAIRS, VISUALISATION, AMBER

Table of Contents

- Phenix News 1
- Crystallographic meetings 2
- Expert Advice
  - Fitting tips #11 – Can a helical DNA basepair be Hoogsteen? 2
- FAQ 5
- Short Communications
  - Characterization of base pair geometry 6
  - Visual representations of internal coordinate restraints: Advantages and limitations 10

Editor

Nigel W. Moriarty, NWMoriarty@LBL.Gov

Phenix News

Announcements

Amber

Amber has been added as an option for the chemical information in a macromolecular refinement in Phenix. Starting with dev-2247, the documentation and code is synced to enable the use of Amber in refinement and geometry minimisation. Furthermore, once Amber has been configured, the Phenix GUI adds an input tab with the appropriate program inputs.

AFITT

In a similar fashion, OpenEye's AFITT program for providing chemical information about ligands has been added to Phenix including documentation and GUI interface.

X-ray diffraction indexing and integration programs DIALS and Xia2 now available in Phenix

Two software packages for the analysis and reduction of X-ray diffraction images are now being included in nightly Phenix builds starting at dev-2307: DIALS (Diffraction Integration for Advanced Light Sources) and Xia2. DIALS is a new implementation of spotfinding, indexing and integration algorithms useful for processing x-ray diffraction data prior to scaling and merging. Created as a collaboration between developers at the Diamond Light Source, members of the Computational Crystallographic Initiative at Lawrence Berkeley National Labs, and members of CCP4, the software provides an extensible framework for algorithms needed for diffraction data reduction and visualization. The software is built on the *cctbx* libraries and implements well-established indexing and integration

methods. Furthermore, its extensible design is providing a platform for the creation of new methods, such as new techniques for indexing multiple, overlapping diffraction patterns, new parameterizations for the refinement of crystallographic models, and new treatments for background and signal during integration. Xia2, developed at the Diamond Light Source, is an expert system designed to automate and simplify diffraction data reduction and merging using, if available on the user's system, Mosflm, Scala, XDS, LABELIT, Aimless, Pointless, and recently, DIALS. Note that Xia2 requires CCP4 as a dependency.

Project websites for tutorials and documentation are available at dials.github.io/ and xia2.github.io/

References

Waterman DG, Winter G, Parkhurst JM, Fuentes-Montero L, Hattne J, Brewster A, Sauter NK, Evans G (2013). "The DIALS framework for integration software." *CCP4 Newsletter on Protein Crystallography* **49**, 16-19.

Winter, G (2010). "xia2: an expert system for macromolecular crystallography data reduction." *Journal of Applied Crystallography* **43**, 186-190.

Acknowledgements

DIALS development at Diamond Light Source is supported by the BioStruct-X EU grant, Diamond Light Source, and CCP4.

DIALS development at Lawrence Berkeley National Laboratory is supported by National Institutes of Health / National Institute of General Medical Sciences grant R01-GM095887. Work at LBNL is performed under Department of Energy contract DE-AC02-05CH11231.

New programs

phenix.AmberPrep

Amber requires two additional files (`.prmtop` and `.rst7`) be supplied to a refinement. In addition, the ordering of the input PDB model file must match order in these files. AmberPrep reads a model PDB file and generates a new model PDB and the two additional files for input into refinement. Ligands with the correct code will also be processed.

New features

phenix.structure_search

- Quickly (~1s) find homologous structures for a given model from included internal database. No network is required.
- Option to compile and output a list of ligands found in structures of identified homologs.
- Option to do Blast search for a given model or a sequence file locally. No network is required.
- Option to use a local PDB mirror for PDB file retrieval.

Crystallographic meetings and workshops

Seventh edition of the Macromolecular Crystallography School 2016, May 25-29

Location: Institute Química-Física Rocasolano CSIC (Madrid, Spain).

2016 Annual Meeting of the American Crystallographic Association, July 22-26

Location: Denver, CO.

The 30th European Crystallographic Meeting, August 28-September 1, 2016

Location: Basel, Switzerland.

Expert advice

Fitting Tip #11 – Can a helical DNA base pair be Hoogsteen?

Bradley Hintze and Jane Richardson, *Duke University*

The nice thing about DNA for a crystallographer is that it is nearly always B-form helix, perhaps with interesting perturbations, and that the G•C and A•T base pairs will have the canonical Watson-Crick (WC) geometry. However, very occasionally a Hoogsteen (HG) base pair is identified, usually in a functionally suggestive place (Aishima 2002; Abrescia 2004; Kitayner 2010). Spin-relaxation NMR studies show quite clearly that normal B-

form base pairs in DNA sample the Hoogsteen arrangement transiently, at populations of about 0.5% and lifetimes of about 1 ms (Nikolova 2011; Alvey 2014). In fact, in the first crystal structure of a DNA base pair, Karst Hoogsteen observed an A•T in the non-WC arrangement that now bears his name (Hoogsteen 1959). That arrangement involves a 180° flip of the purine (A or G) base around the glycosidic bond (from *anti* to *syn*), different H-bond partners, and a shortening of the C1'-C1' distance across the helix (see figure 1). This change is harder for a G•C than for an A•T base pair, because the cytosine N3 must be protonated, and the H-bonds are reduced from 3 in WC to 2 in HG.

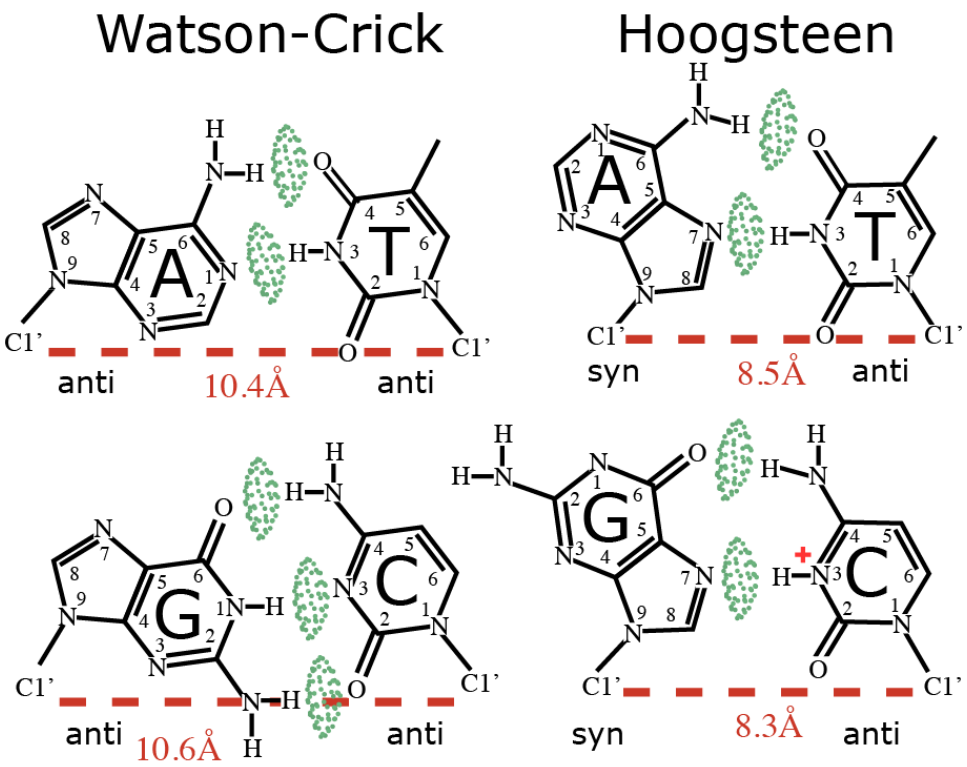


Figure 1: A comparison of the Watson-Crick (WC; left) and Hoogsteen (HG; right) basepairing arrangements. H-bonds are shown by the lens shapes of pale green all-atom contact dots, and the diagnostic distances between C1' atoms are marked in red. Note the plus charge from protonation of the N3 of the cytosine that must happen to enable the less favorable G•C Hoogsteen basepair.

Other contexts for Hoogsteens

Hoogsteen base pairs are a common component of base triples in either RNA or DNA (since after making one Watson-Crick pair the Hoogsteen edge is the next best available for H-bonding). They are not detected, even transiently, in RNA A-form helices, and are now known to be rare but possible in DNA B-form helices.

Occurrence in DNA crystal structures

A recent survey of DNA structures at ≤ 3.0Å resolution found 106 A•T and 34 G•C base pairs that satisfied all three criteria: HG-type H-bonds (each ≤ 3.5Å), *syn* purine χ, and C1'-C1' distance ≤ 9.5Å (Zhou 2014), many of them not mentioned in the accompanying publications. The overall occurrence level of 0.3%, the preference for A•T over G•C, and the preference for TA over AT steps along the sequence, are all quite consistent with the solution NMR and other previous data. New results are an enrichment of HG at helix ends over interiors, a preference for protein or ligand complexes over naked DNA, a preference for the *syn* purine to be at the 5' end vs the 3' end of the helix, and for interior HG pairs to bend the helix slightly toward the major groove side. Although very clear in these crystal structures (e.g. Figure 2) and seen transiently by NMR, there has yet to be NMR evidence of persistent HG base pairs.

Refitting

We have identified a few unambiguous cases, especially in A-tracts, where a base pair modeled as WC but with a shortened C1'-C1' distance can be rebuilt and refined as an HG pair. The new fit has better sterics (H-bonds or clashes), better geometry, and a better fit to the electron density. An example is dA c1 to dT h1 in 3ufd (see Figure 3). Mixtures of WC and HG alternate conformations would also be expected, but of course are harder to verify. We successfully fit such a case for dA 2 of 4auw, where each conformation has a positive difference peak next to the N7, while both conformations together as alternates refine well and lose the difference

density. Note that any WC vs HG rebuild requires significant change in the backbone conformation, especially for the purine.

Conclusion

Our data show that Hoogsteen basepairs are very rare but when they do occur it's mostly in the context of a protein/ligand-DNA complex or at duplex termini. Despite their rarity, it is well worth considering a Hoogsteen basepair when fitting B-form helical DNA. This is especially true if you encounter interpretable difference peaks on either side of the purine, suggestive clashes, or a shortened C1'-C1' distance. As with all refittings, the new fit should improve geometry, stertics, and/or the fit to density. The latter criterion is especially important – Hoogsteen basepairs are extremely rare, so don't do it too often!

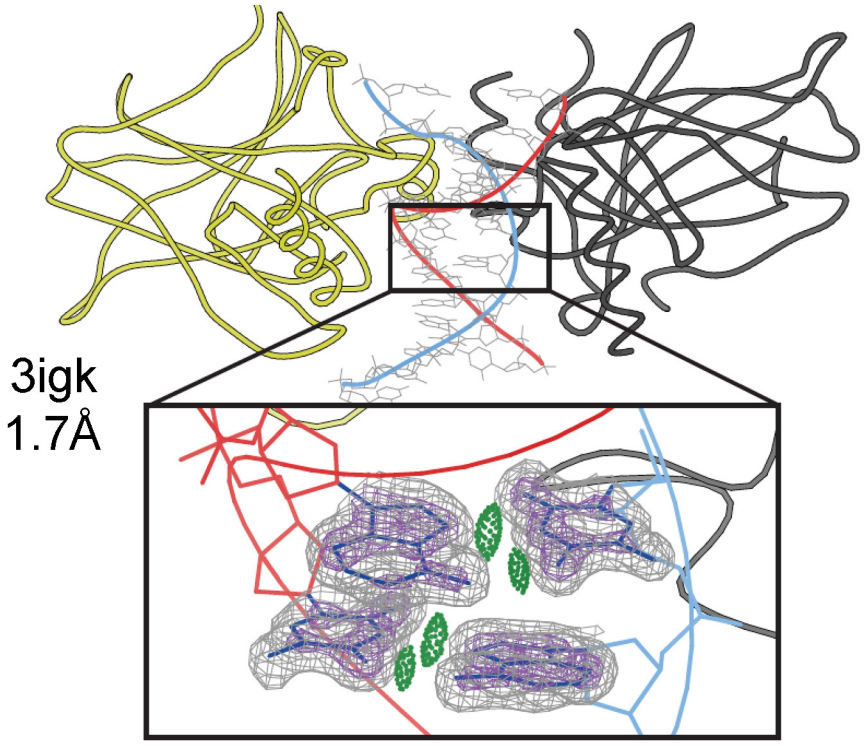


Figure 2: An indisputably correct tandem pair of A•T Hoogsteen base pairs in a complex of p53 protein recognizing an atypical duplex DNA sequence (3igk; Kitayner 2010). From Hintze 2015.

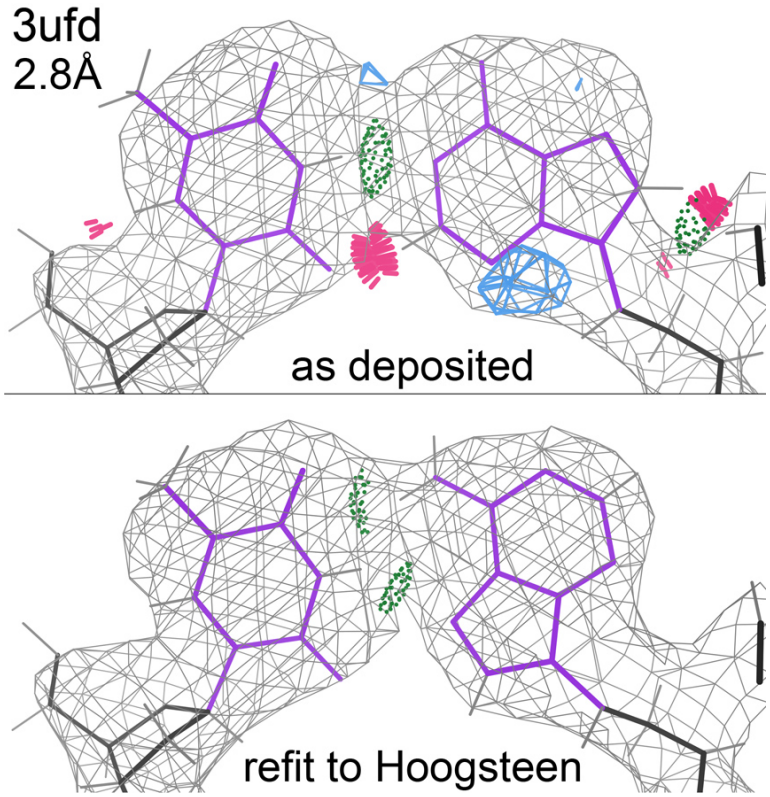


Figure 3: A problematic Watson-Crick base pair at top, with a large clash, distorted base geometry, only one H-bond between the bases, and a positive difference peak (blue) next to the purine N7 (4auw dA h2; Textor 2013). A rebuild that flipped the A base from *anti* to *syn*, adjusted the backbone, and re-refined, produced the much more satisfying fit shown below.

References:

Abrescia NG, Gonzales C, Gouyette C, Subirana JA (2004) "X-ray and NMR studies of the DNA oligomer d(ATATAT): Hoogsteen base pairing in duplex DNA", *Biochemistry* **43**: 4092-4100

Aishima J, Gitti RK, Noah JE, Gan HH, Schlick T, Wolberger C (2002) "A Hoogsteen base pair embedded in undistorted B-DNA", *Nucleic Acids Resl* **30**: 5244-5252

Alvey HS, Gottardo FL, Nikolova EN, Al-Hashimi HM (2014)

"Widespread transient Hoogsteen base pairs in canonical duplex DNA", *Nat Commun* **5**: 4786

Hintze BJ (2015), "Rare Sidechain Conformations in Proteins and DNA," Ph.D. thesis, Duke University.

Hoogsteen K (1959) "The Structure of Crystals Containing a Hydrogen-Bonded Complex of 1-Methylthymine and 9-Methyladenine", *Acta Crystallogr* **12**: 822-823

Kitayner M, Rozenberg H, Rohs R, Suad O, Rabinovich D, Honig B, Shakked Z (2010) "Diversity in DNA recognition by p53 revealed by crystal structures with Hoogsteen base pairs", *Nat Struct Mol Biol* **17**: 423-4292403-240

Nikolova EN, Kim E, Wise AA, O'Brien PJ, Andricoaei I, Al-Hashimi HM (2011) "Transient Hoogsteen base pairs in canonical duplex DNA", *Nature* **470**: 498-502

Textor LC, Holton S, Wilmanns M (2013) "Crystal structure of the bZIP homodimeric Mafb in complex with the C-Mare binding site", unpublished

Zhou H, Hintze BJ, Kimsey IJ, Sathyamoorthy B, Yang S, Richardson JS, Al-Hashimi HM (2015) "New insights into Hoogsteen base pairs in DNA duplexes from a structure-based survey", *Nucleic Acids Resl* **43**: 3420-3433

FAQ

Why can't I see the link I want in my model?

Phenix recently (1.10.1-2155) began writing LINK records into the output model file to facilitate better visualisation of the model used internally by the refinement package including the automatic linking algorithm's results. The second short communication has a discussion of the relationship between a refinement package and a visualisation package.

Phenix, in general, does not use the LINK records in the input model.

Characterization of base pair geometry

Xiang-Jun Lu,[a] & Wilma K. Olson[b]
[a]*Department of Biological Sciences, Columbia University, New York, NY 10027*
[b]*Department of Chemistry and Chemical Biology, Rutgers – The State University of New Jersey, Piscataway, NJ 08854*

Correspondence email: xiangjun@x3dna.org

Introduction

The interactions of the planar nitrogenous bases (A, C, G, T, and U) play a critical role in the three-dimensional organization of DNA and RNA. The relative spatial arrangement of a pair of associated bases can be rigorously quantified by six rigid-body parameters (Dickerson et al., 1989): three translational parameters called Shear, Stretch, and Stagger, and three rotational parameters denoted Buckle, Propeller (twist), and Opening (figure 1). The numerical values of these base pair parameters or the six step parameters used to describe the positioning of neighboring base pairs depend upon the choice of reference frame (Lu et al., 1999). The establishment of a standard base reference frame (Olson et al., 2001) and its implementation in 3DNA (Lu et al., 2003) and Curves+ (Lavery et al., 2009) has largely resolved discrepancies in the analysis of double-helical structures of Watson-Crick (WC) base pair.

The standard reference frame (Olson et al., 2001) is base centric, defined with respect to an 'idealized', perfectly planar WC pair where all six parameters are null. As noted in the classic Dickerson B-DNA dodecamer (Drew et al., 1981), WC pairs are normally non-planar, with Propeller (and Buckle) significantly different from zero. Notably, Propeller in right-handed DNA double helices has a mean value of around -12° (Dickerson et al., 1989; Olson et al., 2001), and its persistence has been rationalized in terms of the increased base-stacking interactions found in these structures (Calladine, 1982; Levitt, 1978). More generally, among the six base pair parameters, Buckle, Propeller, and Stagger describe the *non-planarity* of a pair: Buckle and Propeller render the two bases *non-parallel*, whilst Stagger leads to a vertical separation. On the other hand, Shear, Stretch, and Opening are critical for characterizing different types of non-WC pairs (Lu et al., 2015; Lu et al., 2003). For example, the G–U wobble pair is characterized by a Shear of -2.2 Å; if counted the other way around, i.e., as U–G, the Shear value is +2.2 Å instead.

3DNA (Lu et al., 2003) uses the standard base reference frame (Olson et al., 2001) to calculate six *local* base pair parameters with its 'analyze' routine. The parameters unambiguously characterize any base pair, be it WC, G–U wobble, or non-canonical (e.g., a Hoogsteen pair). Conversely, given the six parameters and base sequence, the 3DNA 'rebuild' routine rigorously reconstructs the spatial disposition of the two interacting bases. This reversibility is one of the unique features of 3DNA, applicable to pairs of bases in both DNA and RNA.

Simple base pair parameters

Although the six *local* base pair parameters are rigorous and serve a good purpose, their numerical values can be cryptic for non-canonical pairs, most notably the values of Buckle and Propeller. Two recent CCN articles (Richardson, 2015a, 2015b) emphasized the importance of base

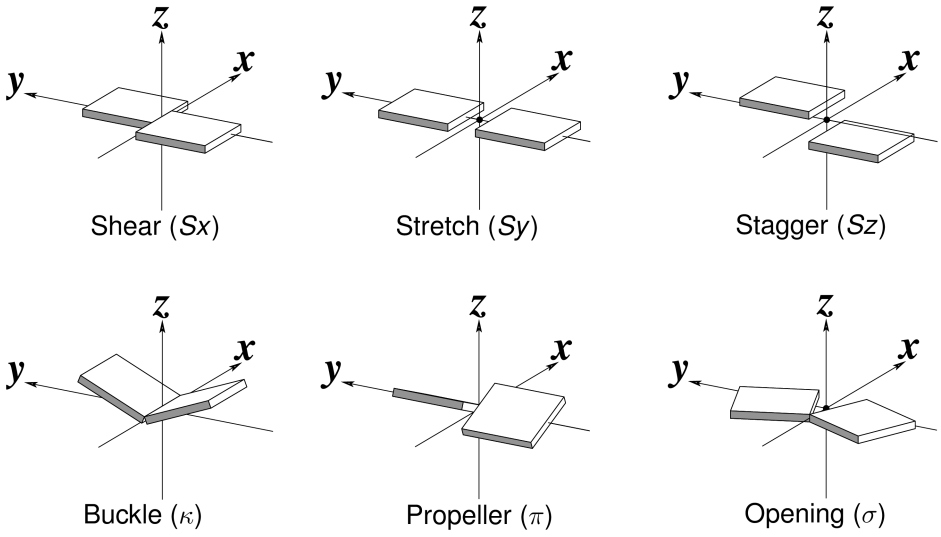


Figure 1: Schematic diagrams of the six rigid-body parameters commonly used for the characterization of base pair geometry.

pair non-planarity at biologically significant positions in high-resolution nucleic acid structures (e.g., functional binding sites) and the need to account correctly for this non-planarity in deriving models of DNA and RNA based on low-resolution data. The account of two of the classic measures of base pair deformations (figure 1) given in the article — "the 'propeller-twist' torsion around a line joining the two bases, and the 'buckle' angle of their bend across the line of base pair H-bonds" — prompted us to derive a new set of six *simple* parameters for a complete qualitative description of base pair geometry. The term '*Simple*' is used because the parameters are more intuitive for non-canonical pairs (figure 2), and to differentiate them from the existing *local* base pair parameters. The *simple* parameters have been implemented in both the 3DNA 'analyze' routine and the new DSSR software (Lu et al., 2015).

Detailed definitions of the *simple* base pair parameters, with worked examples, can be found on the 3DNA homepage (x3dna.org). The key differences between the *simple* and *local* parameters lie in the definition of the base pair coordinate frame and in the description of angular parameters. The *simple* treatment uses the locations of atoms on the interacting purine (R) and pyrimidine (Y) bases, either RC8/YC6 (default) or RN9/YN1, as the (long) ***y***-axis of the pair, corrected to be orthogonal with the ***z***-axis. The ***z***-axis is the average of the *z*-axes of the two bases, following the definition of the corresponding *local* base pair axis and taking the anti-parallel direction into consideration (e.g., in WC pairs). The (short) ***x***-axis fulfills the right-handed rule. The three translational parameters (figure 1) are simply the projections of the vector linking the origins of the two base-reference frames onto the respective ***x***-, ***y***- and ***z***-axes, similar to the definition of the corresponding *local* base pair parameters. The three *simple* angular parameters, however, are defined as 'torsion' angles: Buckle as the 'torsion' angle of the *z*-axes of the two bases with respect to the short base pair axis, Propeller as the 'torsion' angle of the two *z*-axes with respect to the long base pair axis, and Opening as the 'torsion' angle of the two *y*-axes with respect to the base pair ***z***-axis. While the *simple* base pair parameters of WC pairs closely

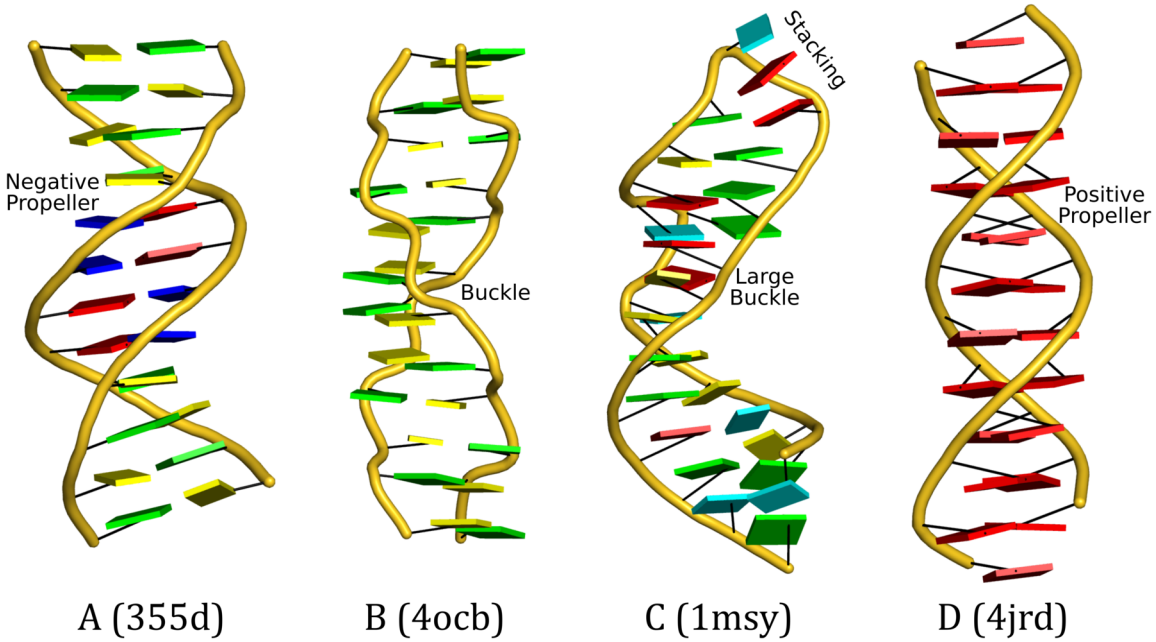


Figure 2: DSSR-introduced cartoon-block representations of DNA and RNA structures that combine PyMOL cartoon schematics with color-coded rectangular base blocks: A, red; C, yellow; G, green; T, blue; and U, cyan. (A) The Dickerson B-DNA dodecamer solved at 1.4-Å resolution [PDB id: 355d (Shui et al., 1998)], with significant negative Propeller. (B) The Z-DNA dodecamer [PDB id: 4ocb (Luo et al., 2014)], with virtually co-planar C–G pairs at the ends, and noticeable Buckle in the middle. (C) The GUAA tetraloop mutant of the sarcin/ricin domain from *E. coli* 23 S rRNA [PDB id: 1msy (Correll et al., 2003)], with large Buckle in the A+C pair, and base-stacking interactions of UAA in the GUAA tetraloop (upper-right corner). (D) The parallel double-stranded poly(A) RNA helix [PDB id: 4jrd (Safaee et al., 2013)], with up to +14° Propeller. The simple, informative cartoon-block representations facilitate understanding of the base interactions in small to mid-sized nucleic acid structures like these. The base identity, pairing geometry, and stacking interactions are obvious.

resemble the *local* base pair parameters, the two sets can differ significantly for non-canonical pairs.

Cartoon-block representations

The new DSSR component of 3DNA (Lu et al., 2015) includes an easy way to create highly effective cartoon-block representations that showcase base pair geometry and base-stacking interactions (figure 2). DSSR has been tested against all nucleic-acid-containing structures in the Protein Data Bank (PDB), and works with atomic coordinate files in either PDB or PDBx/mmCIF format. Figure 2 illustrates the arrangements of bases in four representative high-resolution X-ray crystal structures. The following command generates the script, named '`355d.pml`', needed to construct figure 2A (the Dickerson B-DNA dodecamer solved at 1.4 Å resolution, PDB id: 355d)within PyMOL:

```
x3dna-dssr -i=355d.pdb -o=355d.pml\
    --cartoon-block
```

Once '`355d.pml`' is loaded into PyMOL, the commands '`orient; turn z, -90; ray`' set the structure in the most extended view vertically and perform the ray-tracing needed to obtain the high-resolution PNG image. Similar commands apply to the other three cases. Details on reproducing the illustrated images, and outputs (including the *simple* base pair parameters) of DSSR and the 3DNA 'analyze' program for the four structures, are available at: x3dna.org/highlights/CCN-on-base-pair-geometry.

In addition to the default style shown in figure 2, DSSR provides several variations of the cartoon-block representation of planar, aromatic bases. For example, one can change both the thickness and the sizes of the blocks, shade the minor-groove edges of the bases, and represent a WC pair as a single long rectangular base pair block. One can also orient a structure in a specific base or base pair reference frame for easy comparison, and can attach the helical axes to an array of stacked base pairs. See the x3dna.org for more examples.

Summary

Base pair geometry can be described in different ways. The existing set of six *local* base pair parameters in 3DNA is mathematically rigorous, allowing for an unambiguous characterization of any pair of interacting bases and serving as input for exact model building. The new set of six *simple* base pair parameters described herein provides a more intuitive interpretation of intra-base pair structural variations, especially for the out-of-plane Buckle and Propeller distortions of non-canonical base pairs. The two sets of base pair parameters complement one another and serve different audiences and/or purposes. Numerical values of both sets of parameters are readily obtained with 3DNA. Moreover, the DSSR component of 3DNA provides highly effective cartoon-block representations of the base interactions within a nucleic acid structure.

References

Calladine, CR. (1982). Mechanics of sequence-dependent stacking of bases in B-DNA. *J Mol Biol, 161*(2), 343-352.

Correll, CC, & Swinger, K. (2003). Common and distinctive features of GNRA tetraloops based on a GUAA tetraloop structure at 1.4 Å resolution. *RNA, 9*(3), 355-363.

Dickerson, RE, Bansal, M, Calladine, CR, Diekmann, S, Hunter, WN, Kennard, O, von Kitzing, E, Lavery, R, Nelson, HCM, Olson, WK, Saenger, W, Shakked, Z, Sklenar, H, Soumpasis, DM, Tung, C-S, Wang, AH-J, & Zhurkin, VB. (1989). Definitions and nomenclature of nucleic acid structure parameters. *EMBO J, 8*(1), 1-4.

Drew, HR, Wing, RM, Takano, T, Broka, C, Tanaka, S, Itakura, K, & Dickerson, RE. (1981). Structure of a B-DNA dodecamer: conformation and dynamics. *Proc Natl Acad Sci U S A, 78*(4), 2179-2183.

Lavery, R, Moakher, M, Maddocks, JH, Petkeviciute, D, & Zakrzewska, K. (2009). Conformational analysis of nucleic acids revisited: Curves+. *Nucleic Acids Res, 37*(17), 5917-5929.

Levitt, M. (1978). How many base pairs per turn does DNA have in solution and in chromatin? Some theoretical calculations. *Proc Natl Acad Sci U S A, 75*(2), 640-644.

Lu, XJ, Bussemaker, HJ, & Olson, WK. (2015). DSSR: an integrated software tool for dissecting the spatial structure of RNA. *Nucleic Acids Res, 43*(21), e142.

Lu, XJ, & Olson, WK. (1999). Resolving the discrepancies among nucleic acid conformational analyses. *J Mol Biol, 285*(4), 1563-1575.

Lu, XJ, & Olson, WK. (2003). 3DNA: a software package for the analysis, rebuilding and visualization of three-dimensional nucleic acid structures. *Nucleic Acids Res, 31*(17), 5108-5121.

Luo, Z, Dauter, M, & Dauter, Z. (2014). Phosphates in the Z-DNA dodecamer are flexible, but their P-SAD signal is sufficient for structure solution. *Acta Crystallogr D Biol Crystallogr, 70*(Pt 7), 1790-1800.

Olson, WK, Bansal, M, Burley, SK, Dickerson, RE, Gerstein, M, Harvey, SC, Heinemann, U, Lu, XJ, Neidle, S, Shakked, Z, Sklenar, H, Suzuki, M, Tung, CS, Westhof, E, Wolberger, C, & Berman, HM. (2001). A standard reference frame for the description of nucleic acid base pair geometry. *J Mol Biol, 313*(1), 229-237.

Richardson, JS. (2015a). A context-sensitive guide to RNA & DNA base pair & base-stack geometry. *Computational Crystallography Newsletter, 6*(Part 2), 47-53.

Richardson, JS. (2015b). Fitting Tip #10 – How do your base pairs touch and twist? *Computational Crystallography Newsletter, 6*(Part 2), 28-31.

Safaee, N, Noronha, AM, Rodionov, D, Kozlov, G, Wilds, CJ, Sheldrick, GM, & Gehring, K. (2013). Structure of the parallel duplex of poly(A) RNA: evaluation of a 50 year-old prediction. *Angew Chem Int Ed Engl, 52*(39), 10370-10373.

Shui, X, McFail-Isom, L, Hu, GG, & Williams, LD. (1998). The B-DNA dodecamer at high resolution reveals a spine of water on sodium. *Biochemistry, 37*(23), 8341-8355.

Note added in proof

The DSSR cartoon-block representation (figure 2) has been integrated into PyMOL via the 'dssr_block' command (http://pymolwiki.org/index.php/Dssr_block).

Visual representations of internal coordinate restraints: Advantages and limitations

Pavel V. Afonine and Nigel W. Moriarty

Department of Biophysics and Integrated Bioimaging, Lawrence Berkeley Laboratory, Berkeley, CA 94720

Correspondence email: NWMoriarty@LBL.Gov

Macromolecular structure refinement can be thought of as an optimization problem with the goal to obtain a model that describes the experimental data as well as possible. Wikipedia defines optimization as:

> In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics. More generally, optimization includes finding "best available" values of some objective function given a defined domain (or a set of constraints), including a variety of different types of objective functions and different types of domains.

In case of protein crystallography, there is a model typically consisting of two components: atomic model (protein, DNA/RNA, ordered solvent and ligands) and non-atomic model, such as bulk solvent. There is also experimental data consisting of intensities of diffracted light (X-ray or neutron, for instance). The model is changed systematically to better fit the experimental data.

In other fields such as cryo-electron microscopy (cryo-EM), the model is usually atomic model as well and the data are the 3D reconstruction volume map. Various methods can be used for optimization of the model into the map: from rigid-body docking and flexible fitting to gradient-driven minimization and simulated annealing.

What's common to macro-molecular crystallography and cryo-EM is that experimental data is almost always of insufficient quality to refine parameters of atomic model (coordinates, for example) individually. Therefore to make refinement practical restraints or constraints are used with the corresponding refinements called restrained or constrained refinement. In what follows we focus on restrained refinement. In restrained refinement the target function contains two components

$$T = T_{data} + \omega * T_{restraints}$$

with T_{data} describing how well the model fits the data and $T_{restraints}$ being a source of extra *a priori* knowledge about the molecule added with some weight. This knowledge is the information about covalent bond lengths and angles, dihedral angles, planar molecules (such as phenylalanine ring), chiral centers and nonbonded interactions:

$$T_{restraints} = T_{bond} + T_{angle} + T_{dihedral} + T_{plane} + T_{chiral} + T_{nonbonded}$$

Sometimes even this information isn't sufficient and more is needed. This may be, for example, information about secondary structure arrangements of proteins or nucleic acid molecules, distribution of main chain conformations (Ramachandran plot) or side chain conformations (rotamers). Similar to above, this information is added as extra terms to $T_{restraints}$.

Typically, terms in $T_{restraints}$ are sums of squared differences between values found in current model and "library" values, for

example:

$$T_{bond} = \sum_{\text{pairs of covalently bound atoms}} \frac{1}{\sigma^2}\left(d_{model}-d_{library}\right)^2$$

The library values are tabulated values collected from various sources such as derived from high-quality high-resolution small molecules, spectroscopy experiments and theoretical calculations that are compiled into libraries such as CCP4 Monomer Library (Vagin et al. 2004) or GeoStd (see Notes). Recent developments in restraint libraries include dynamic restraints based on the conformation of the protein backbone (Moriarty et al. 2014).

Refinement packages routinely apply the restraints from the libraries and, in addition, will automatically apply a set of links to provide a more complete chemical picture of the macromolecule. Examples are the peptide link and disulfide bridge. The former is generally based in the order of the protein residues and other criteria such as distance, residue type and/or atom names. The disulfide link is generally based on proximity. In either case a standard link can be applied from the library to create the bonding although Phenix now uses new link information that includes handedness (Sobolev et al. 2015). The library of standard links extends to a subset of the possible carbohydrate links, either protein-carbohydrate or intra-saccharide.

It is not uncommon that a structure may contain novel ligands that are covalently linked to the macromolecule or each other. The novel ligands will often require restraints be generated to add to the restraints target function. It is not unreasonable to expect that the covalent links be non-standard and unique enough to not be present in existing libraries or outside the scope of automatic linking procedures. This means that in order to make a refinement program aware of such unusual bonds (so that the program adds corresponding term to T_{bond}) a user needs to convey the program such information. This can be done by a variety of ways such as using atom selection syntax to specify custom bonds; or creating a link in the same format as the library and specifically applying the link, which allows for more complexity.

A feature of many of the Phenix programs is the ability to write a file that contains all the restraint information used by the program. To ascertain the specified link was actually used in refinement one can inspect `.geo` file that `phenix.refine` creates. The geometry restraints are grouped into restraint types. A typical bond section is shown in schema 1. Each bond is listed in order of decreasing residual and contains the ID string of each atom, the ideal bond length from the library, the value of the bond in the model, the

```
Bond restraints: 4714
Sorted by residual:
bond pdb=" C2D HEM A 201 "
     pdb=" C3D HEM A 201 "
  ideal  model  delta    sigma   weight residual
  1.334  1.521 -0.187 2.00e-02 2.50e+03 8.74e+01
bond pdb=" CA  VAL A  14 "
     pdb=" CB  VAL A  14 "
  ideal  model  delta    sigma   weight residual
  1.537  1.563 -0.025 1.29e-02 6.01e+03 3.88e+00
```

Schema 1: The first three bond restraints of a typical geometry file including a ligand (HEM). Each bond lists the atom ID strings, ideal and actual values, the difference, sigma, weight and residual.

difference of the ideal and actual, the sigma, weight and residual or contribution to T_{bond}. The file is written at the beginning of a refinement and can optionally be written at the end of a refinement, which is useful to check how close the model approaches the restraints. A simple program is available, `elbow.refine_geo_display` that can be used to display restraints of an atom selection.

Visualisation of the refinement model is a powerful tool allowing greater understanding of the model compared to a list of restraints in a `.geo` disk file. However, it is the file that contains the absolute information. When displaying molecules using graphic programs (such as Coot (Emsley et al. 2010) or PyMOL (DeLano 2002)) representations of the bonds in a model are drawn on the screen based on a number of different criteria. The criteria differ between programs and version. Links between entities such as carbohydrate units, novel ligands and metals are especially difficult to represent as the graphics program does not know what the refinement package did internally and can only rely on the information in the model file. These links may not be drawn by the graphic program as a bond (solid or dashed stick connecting two atoms involved), which may result in confusion and suspicion that the bond in question was not used in refinement.

For example, the linking of NAG to ASN is a common glycosidic bond in proteins. When displaying this region of the protein Coot uses the LINK record in the PDB to draw the glycosidic bond between the ASN and NAG (see figure 1a) using a dashed line. Removing the LINK record results in the visualization of the bond being absent (see figure 1b). As stated earlier, the sequence of the entities can lead to linking. In the case of Coot, a PDB with just the ASN and NAG entities results in a solid line representing the bond between the two. Furthermore, changing the code of NAG to LIG and inserting a TER card between the two entities gives the visualisation in figure 1c — a

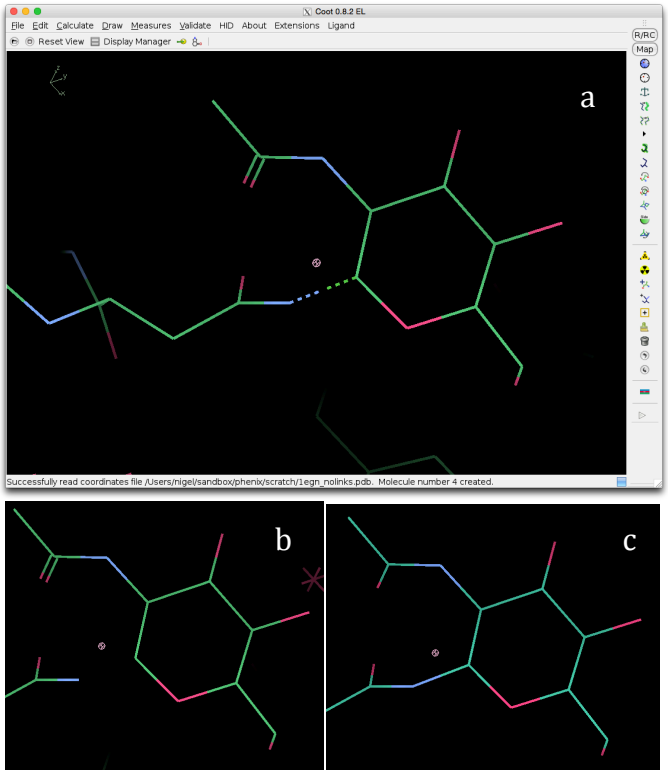


Figure 1: Comparison of the visualization of a glycosidic bond between ASN and NAG with LINK record (a), without LINK record (b) and as a pair of sequential entities (c).

solid line drawn by Coot between the two closet atoms in sequential entities. The visualisation of the ASN-NAG link is unchanged between the entities regardless of whether the link is being specific explicity specified (LINK) or implied (NAG is usually linked to ASN) or explicitly excluded (TER card).

When visualising the same model in PyMOL 0.99, the link is based on distance and ignores completely the identity of the entities. This means that in all cases spoken about in the previous paragraph, there is a solid line between the ASN and NAG representing the glycosidic bond (see figure 2). It should be noted that version 0.99 is quite old and the newer version may behave differently.

Displaying bond valence is also an important part of the visualisation of the model. PyMOL

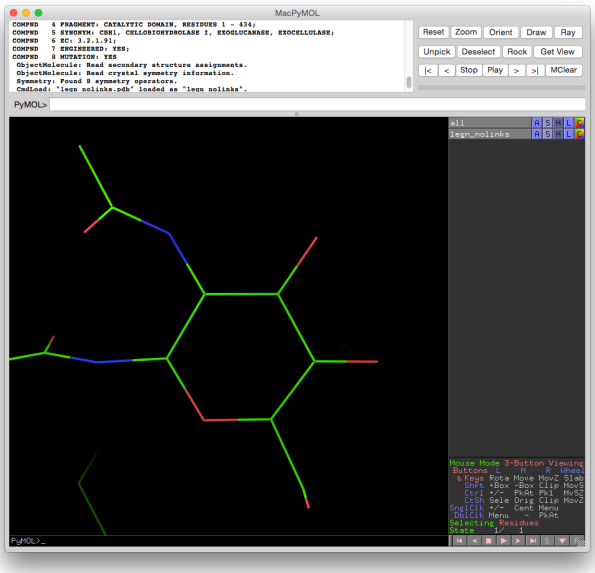


Figure 2: PyMOL visualisation of the NAG in 1EGN.

has a setting that defaults to false. To see the double bond in PyMOL, the command "`set valence, 1`" is required to enable it. The determination of the bond does not rely on the residue code. In the ASN-NAG example, changing the NAG code to LIG does not change the depiction of the double bond as two lines.

An example of when bond valence visualisation can be helpful is p-coumaroyl-shikimate bound to the PvHCT2a protein and deposited in the PDB as 5FAL. There were some important questions that revolved around the bond valence of a particle ring in the ligand. The ring in question (see figure 3) has a single double bond and two chiral centres. The location of the double bond is important for the chemistry of the ligand and mechanism as well as the generation of

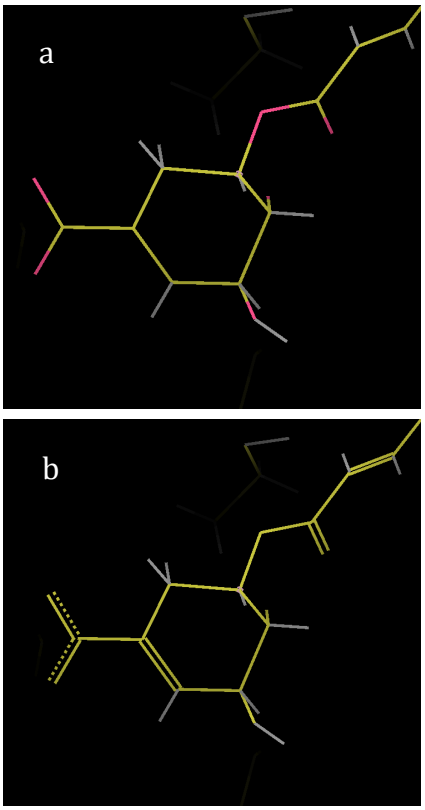


Figure 3: Comparison of a Coot visualization of a novel ligand without the restraints library loaded (a) and with the restraints library loaded (b).

restraints. If the hydrogens had been absent, the visualisation in figure 3a would have been information poor. Loading the novel ligands restraints library into Coot provides the view in figure 3b that is much more informative.

Quick Summary

Visualisation packages do not always know what a refinement package is doing internally and the representation of a model is limited by the data exchange medium.

Notes

GeoStd, An open-source restraints library, http://sourceforge.net/projects/geostd

Wikipedia, The Free Encyclopedia, s.v. "Mathematical optimization" (accessed 26th January, 2016), http://en.wikipedia.org/wiki/Mathematical_optimization

References

DeLano, Warren L. 2002. *PyMOL 0.99*.

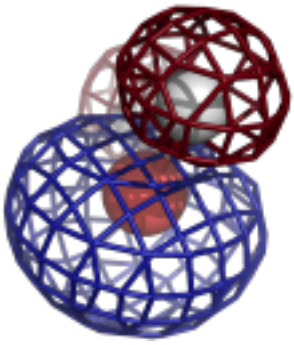
Emsley, P., B. Lohkamp, W. G. Scott, and K. Cowtan. 2010. "Features and Development of Coot." Acta Cryst. D 66: 486–501.

Moriarty, Nigel W., Dale E. Tronrud, Paul D. Adams, and P. Andrew Karplus. 2014. "Conformation-Dependent Backbone Geometry Restraints Set a New Standard for Protein Crystallographic Refinement." *FEBS Journal* 281 (18): 4061–71. doi:10.1111/febs.12860.

Sobolev, Oleg V., Nigel W. Moriarty, Pavel V. Afonine, Bradley J. Hintze, David C. Richardson, Jane S. Richardson, and Adams, Paul D. 2015. "Disulfide Bond Restraints." *Computational Crystallography Newsletter* 6 (1): 13–13.

Vagin, A. A., R. A. Steiner, A. A. Lebedev, L. Potterton, S. McNicholas, F. Long, and G. N. Murshudov. 2004. "REFMAC5 Dictionary: Organization of Prior Chemical Knowledge and Guidelines for Its Use." *Acta Crystallographica Section D-Biological Crystallography* 60 (December): 2184–95. doi:10.1107/S0907444904023510.

VOLUME SEVEN



COMPUTATIONAL CRYSTALLOGRAPHY NEWSLETTER

JULY MMXVI

POLDER, HDF5, BETA STRANDS, MD WATER

Table of Contents

- Phenix News 15
- Crystallographic meetings 16
- Expert Advice
  - Fitting tips #12 – Twist Tells: better β strands at >3.5Å in x-ray or cryoEM 16
- FAQ 20
- Short Communications
  - Using Molecular Dynamics Simulations to Enrich the Water Structure in Biomolecular Crystals 21
- Article
  - Working with EIGER data 25
  - Processing XFEL data with cctbx.xfel and DIALS 32

Editor

Nigel W. Moriarty, NWMoriarty@LBL.Gov

Phenix News

Announcements

Amber

A new version of Amber (ambermd.org) was released 30th April 2016. This version, known as Amber16 (and AmberTools16), has been integrated into Phenix from dev-2499. Please use the documentation from the current installation to install (see FAQ page 20).

New programs

phenix.polder

`phenix.polder` calculates OMIT maps by preventing the bulk solvent mask from penetrating the region of the OMIT atom selection. This tool is useful in cases where the density of the selected atoms is weak and possibly obscured by bulk solvent. Polder maps are less biased than procedures where the atoms are simply removed from the model or where the selected atoms are included in the solvent mask calculation and their occupancy is set to zero. As a larger volume is excluded from the bulk solvent, misinterpretation of bulk solvent density as OMIT density can be avoided. Polder maps are suitable for parts of the structure with weak density (such as ligands, flexible residues, alternative conformations, loop regions). `phenix.polder` is available from the command line and in the Phenix GUI.

New features

Parameters of NCS search procedure

Since the last *Phenix* release (1.10-1) the parameters for non-crystallographic symmetry (NCS) search procedure have been changed. In the current nightly builds the

parameters in the `ncs_search` scope are (with default values)

```
enabled = False
exclude_selection = "not (protein or
nucleotide) or element H or element D"
chain_similarity_threshold = 0.85
chain_max_rmsd = 2.0
residue_match_radius = 4.0
```

The first parameter, `enabled`, turns on NCS search while `exclude_selection` is the selection to choose the part of a model from exclusion – NCS search will disregard the selection in the initial model in the search procedure. By default NCS is searched only in protein or nucleotide chains excluding all ligands, waters and hydrogens. A list chain pairs is generated and checked for similarity using the following criteria:

- `chain_similarity_threshold`: alignment of residues in prospective chains
- `chain_max_rmsd`: RMSD between atoms of superposed chains
- `residue_match_radius`: maximum allowed distance difference between pairs of matching atoms of two residues

If all the criteria are satisfied a chain pair is considered NCS-related.

More relaxed parameters are used for validation of user-supplied NCS groups. If one wishes to supply poorly related NCS groups, it is necessary to relax these parameters even more to ensure the user-supplied definitions pass the validation.

The same procedure and parameters are used to match chains of reference model with chains of refined model.

Additional or updated information can be found in documentation about *phenix.refine* and *phenix.simple_ncs_from_pdb*. See FAQ later in this newsletter about how to access documentation for new features.

cis/*trans* peptide bond geometry control

All geometry-based programs in Phenix can now have the peptide bond specified as *cis* or *trans* regardless of the input geometry. The `phil` parameter scope is useful when are peptide has adopted an incorrect local minimum and needs coërcion to move to the correct confirmation.

Peptide plane control

Refinement of low-resolution structures has many challenges arising from the lack of data. One consequence is the increased number of ω angle outliers. Phenix now has a parameter that will impose a plane restraint on each peptide bond including the C_α from each residue and the intervening C, N and O atoms. This should not be used in the mid- to high-resolution ranges as there is significant deviation in the ω angle from planar as investigated by Karplus *et al*.

Crystallographic meetings and workshops

The 30th European Crystallographic Meeting, August 28-September 1, 2016

Location: Basel, Switzerland.

Expert advice

Fitting Tip #12 - Twist Tells: better β strands at >3.5Å in x-ray or cryoEM

David Richardson and Jane Richardson,
Duke University

β sheets are just as important as α helices, and much more elegant. However, somewhere around 3.5 to 4Å resolution (a range common in the currently most exciting cryoEM and crystal structures) β strands start to merge together patchily in the density and then become a continuous slab. In the *de novo* trace of a new protein even when the helices are very clear, determining orientation and connectivity of β strands is still very difficult or even impossible (Baker 2012). For this purpose, a recent tool called StrandTwister (Si & He 2014) makes use of a powerful source of information apparently new to current automated methodology: the handedness and magnitude of β sheet twist.

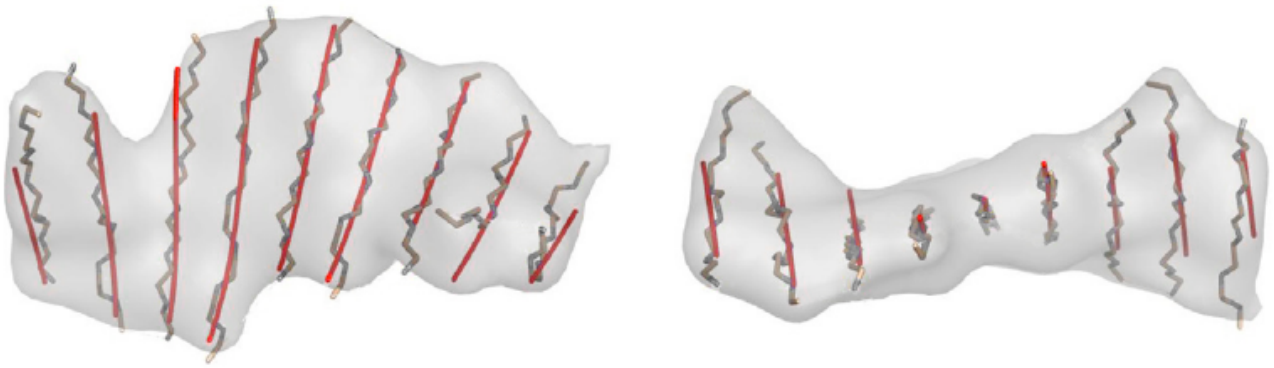


Figure 1: StrandTwister β-strand detection from simulated density maps at 10 A°. The best of the top ten sets of detected β traces (red lines) are superimposed with the backbone of the β strands (dark gray) and the density map (pale gray). A large β sheet in 1UD9_B is shown in top view (left) and in side view, which has maximum right-handed twist (right). (This is a simpler version of Figure 4C from Si & He 2014)

A new tool for cryoEM

The StrandTwister software starts from a slab of density identified as probable β sheet by one of many existing methods (Kong 2003; Baker 2007; Si 2012), but which does not show clearly separated strands. It first fits a polynomial surface to the voxels in the density slab. Then it fits several potential models, with varying β-strand number and orientation, to the polynomial surface. Most importantly, it scores each of the models by the strength of its average right-handed strand twist. The algorithm produces quite accurate tracings within the top-scored few, both within simulated density (figure 1) and within initial experimental maps in the 3 to 7Å resolution range (figures 5-7 in Si & He 2014), and for either parallel or antiparallel β. However, the choice among similar-scoring models, and the connectivity of the strands with the rest of the structure, needs to be made in the context of the full molecular tracing. Executable software for StrandTwister and related information are available at (see footnote 1).

Old advice for crystal structures

StrandTwister is aimed at application to cryoEM structures but it could also be used for low-resolution x-ray. Indeed, a simpler version of the twist criterion was proposed by us in the early stages of protein crystallography, when all initial chain tracing was done manually. Figure 2 (from Richardson 1985) shows the choice of vertical vs horizontal β strands, based on which diagonal pair of density-slab corners twist up vs down. This builds on the long-known rule that β strands always show a right-handed twist as measured by peptide orientation along the strand (Chothia 1973), of between about 0° and 30° per residue (Richardson 1981). If twist were measured perpendicular to the strands, it would give a left-handed value, which is why a given slab of twisted electron density is compatible with only a narrow range of strand orientations. This simple fact can help guide either manual or automated model-building for either x-ray or cryoEM, or can be brought in by using StrandTwister. For each near-optimum strand orientation, the strand number can be estimated from the density-slab width perpendicular to the strands (figure 2).

The early papers also provide further rules that could be of use in current low-resolution modeling, such as which irregularities are frequent and which are vanishingly rare (figure 3). StrandTwister does not yet treat β barrels, but sorting out their strands could

[1]Strandtwister wedbsite – http://www.cs.odu.edu/_jhe/software/strandtwister

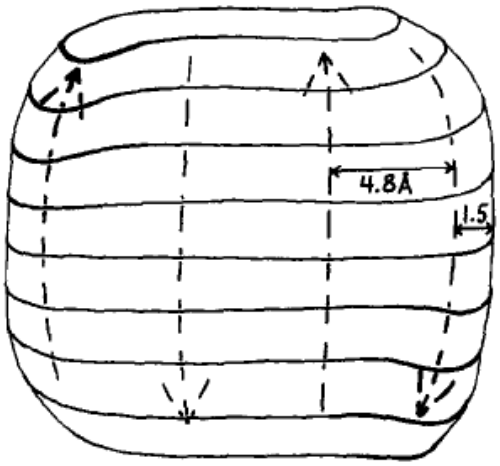


Figure 2: Illustration of how to determine the probable number and direction of b-strands from the width and twist of low-resolution electron density for a b-sheet. If the upper left and lower right corners twist forward (as above), then the strands are vertical, while if the other two corners twisted forward the strands would have to be horizontal. The total width of the sheet of density in a direction perpendicular to the strands should be approximately 4.8(n-1) + 3 Å. (This is hand-drawn Figure 5 from Richardson 1985, with its original caption.)

also be helped by rules from the early lore. Twist matters even in barrels, because it determines the offset between top and bottom of a strand, and thus how its ends can connect with other structure. Parallel barrels in soluble proteins are nearly always 8-strand "TIM barrels", and they are surrounded by a ring of connecting loops nearly all of which contain a recognizable α-helix; both strands and helices make about a 40° angle with the barrel axis.

Antiparallel β barrels are more difficult, because their shapes vary and their overall cross-section changes only slowly with strand number. However, if strand number can be guessed from predictions or from the overall trace, they can then be fairly well placed using empirically observed correlation of shape, twist and strand number. These relationships can be visualized in terms of the strand-crossing angle between front and rear sides of the barrel, as seen in figure 4. Cylindrical barrels, with some β-sheet H-bonding all the way around, average a strand-crossing angle of about 90° for 6 strands (figure 4a), and don't get much below ~70° even for 10 or 11 strands (e.g., in the "can" of green fluorescent protein). If H-bonding is continuous around one end but open on the other, which splits open one end of the cylinder of density as seen for superoxide dismutase in figure 4 of Richardson 1985, the strand-crossing angle is usually near 40° (figure 4b). If β-sheet H-bonding is broken on both sides, to form a β sandwich, then the two sheets can be analyzed separately, but it may help to know that the strand-crossing angle between them is seldom much less than 30° (figure 4c).

The bottom line

Don't give up if your density map has seemingly featureless slabs of density for its β

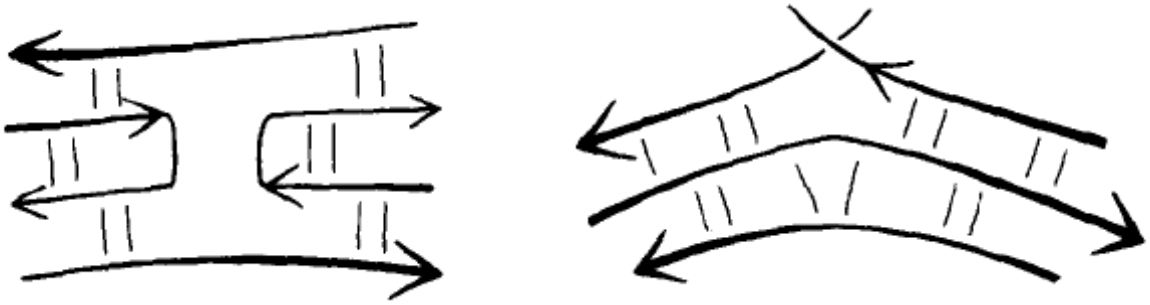


Figure 3: At left is a type of interrupted β-strand that almost always turns out to represent an incorrect tracing (the middle strands should continue straight across). At right is a legitimate sort of interrupted strand, in which the two top segments twist strongly and are not at all colinear. (Figure 8 of Richardson 1985, and its original caption)

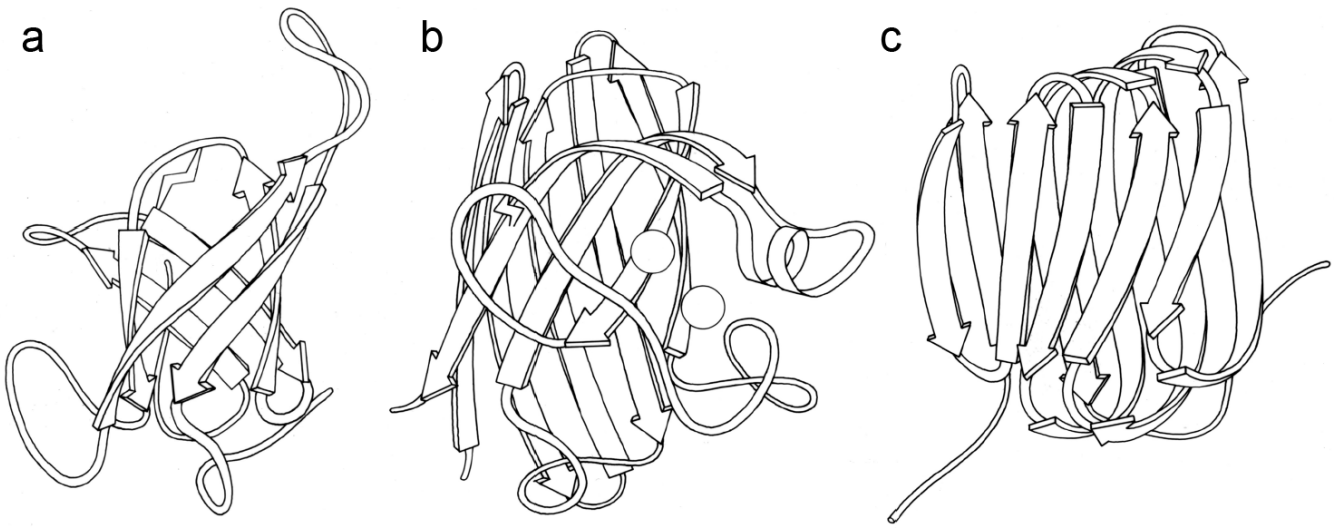


Figure 4: Strand-crossing angles between front and back sides of antiparallel β barrels. (a) The 6-strand, fully H-bonded barrel in trypsin has a strand-cross angle of 90°. (b) The 8-strand barrel in Cu,Zn superoxide dismutase, with one opening between strands, has a strand-cross angle of 60°. (c) The 10-strand β sandwich in tomato bushy stunt virus capsid has a strand-cross angle of 30°.

sheets. Leveraging information from the twist of that density, in addition to its size and shape, may well determine the number and orientation of the β strands quite closely.

References

Baker ML, Ju T, Chiu W (2007) "Identification of secondary structure elements in intermediate-resolution density maps", *Structure* **15**: 7–19

Baker MR, Rees I, Ludtke SJ, Chiu W, Baker ML (2012) "Constructing and validating initial Cα models from subnanometer resolution density maps with pathwalking", *Structure* **20**: 450–463

Chothia C (1973) "Conformation of twisted beta-pleated sheets in proteins", *J Mol Biol* **75**: 295–302

Kong Y, Ma J (2003) "A structural-informatics approach for mining β sheets: locating sheets in intermediate-resolution density maps", *J Mol Biol* **332**: 399–413

Richardson JS (1981) "The anatomy and taxonomy of protein structures", *Adv Prot Chem* **34**: 167–339

Richardson JS, Richardson DC (1985) "Interpretation of electron density maps", *Meth Enzymol* **115**: 189-206

Si D, Ji S, Nasr KA, He J (2012) "A machine learning approach for the identification of protein secondary structure elements from electron cryo-microscopy density maps", *Biopolymers* **97**: 698–708

Si D, He J (2014) "Tracing beta strands using StrandTwister from cryoEM density maps at medium resolutions", *Structure* **22**: 1-12

FAQ

Why is the documentation on the website different from what I'm seeing in my installation?

Major numbered versions of Phenix are released periodically to provide a stable highly tested installations. There are also nightly builds of Phenix (and some external programs like Amber, AFITT and Rosetta) so test the software suite on several operating systems. This provides the ability to release very up-to-date versions to disseminate new programs and features as well as bug fixes.

Documentation generation is part of this nightly testing so that current information can be packaged with each version's installer and accessed via the GUI or typing

```
phenix.doc
```

on the commandline.

The documentation on the Phenix website is associated with the major release. If you have a nightly build or wish access to a feature added since the last major release, you will need to access the documentation from a nightly build. This requires installing the latest nightly build and reading the documentation on your computer.

Using Molecular Dynamics Simulations to Enrich the Water Structure in Biomolecular Crystals

Irem Altan[1], Pavel V. Afonine[2] and Patrick Charbonneau[1]
[1]*Department of Chemistry, Duke University, 124 Science Drive, Box 90346, Durham, NC 27708, USA.*
[2]*Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley CA 94720, USA*

Introduction

Describing the solvation of biomolecules is challenging due to their complex surface geometry with a mosaic of hydrophobic and hydrophilic regions, as well as the intrinsically probabilistic nature of solvation itself. The problem also carries over to crystals of biomacromolecules, which contain between 20 and 90% solvent by volume with an average of about 50% (Weichenberger et al., 2015). The crystal solvent is traditionally considered in two categories: ordered and disordered (or bulk) solvent (figure 1). Solvent at the biomolecule surface constitutes the ordered part and can be probed by diffraction. This portion is modeled using an atomic model to describe distinctly resolved features in the residual Fourier F_o-F_c map. Solvent distant from the protein surface is disordered and cannot be visualized as individual molecules. The disordered part is modeled as a region of flat electron density. Clearly, these two descriptions are overly simplistic and address only two opposite extremes of a spectrum. The reality, of course, is more complex: bulk-solvent density may vary locally (Burling et al., 1996, Lounnas et al., 1994; Sonntag et al., 2011) and the signal arising from semi-ordered solvent may be at or below the noise level in F_o-F_c map. It is therefore not surprising that even high quality X-ray structures of biomolecules can have R-factors as high as 20%; a significant component of this discrepancy may be due to the imperfections in the solvent model (Holton et al., 2014). Further improvement of crystal structure solvent description may be beneficial for:

1. providing higher-quality structural models,
2. assessing models routinely used for simulating water and
3. revealing information that is not directly present in the X-ray diffraction patterns, such as protonation states.

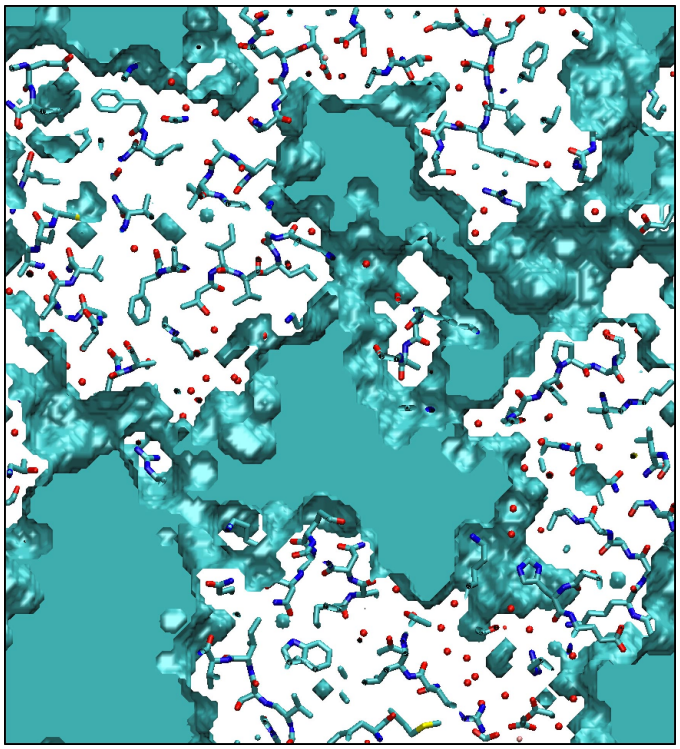


Figure 1: Protein unit cell showing the protein and the water model. Bulk solvent region is shown in blue while red spheres denote crystal waters.

Molecular dynamics (MD) simulations' natural capability to treat arbitrary surfaces and the probabilistic nature of solvation has inspired us to develop a methodology to explore and eventually exploit these strengths to complement existing solvent descriptions. Here we document the work in progress in its rather initial stage.

MD simulations

The electron density obtained from diffraction data is an average over both unit cells and time, with each configuration contributing to the diffraction proportionally to its occurrence. For an ergodic fluid, the time

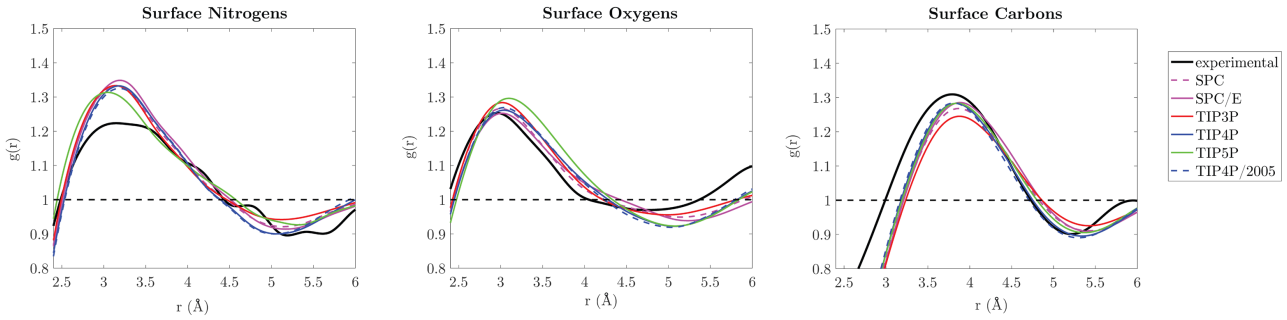


Figure 2: Radial distribution functions, calculated by averaging electron density in shells extending from the protein surface, for surface nitrogens, oxygens, and carbons. (Shown for experimental densities (black) and various water models).

distribution of water molecules within a given unit cell is equal to their instantaneous distribution in all unit cells. Hence, peaks to which crystal waters are typically assigned correspond to a high occupancy probability. Because that occupancy can be far below unity, approaches that can sample it more systematically can be potentially far-reaching. MD simulations, for instance, provide a naturally probabilistic description of solvation. Its biggest weakness is that the probability distribution it actually samples may differ from that observed in experiments, because it relies on imperfect (classical) force fields. In order to examine how well it fares, we started simulating water and protein within the crystal unit cell. Averaging over only the water density in the MD snapshots (keeping the biomolecule fixed) yields an estimate of the solvent distribution in the unit cell.

We use a test protein, (PDB ID: `1YTT`), for which experimental phases are available, allowing us to probe the solvent density without bias from the protein model (Burling et al., 1996). The simulated system uses a protein in its unit cell with water molecules from different water models used to describe water in MD simulations, in order to assess their ability to reconstruct the solvent density. We use `Gromacs` (Berendsen et al., 1995) to run MD simulations. For each MD snapshot, we generate model-calculated electron densities using `Phenix` (Adams et al., 2010). We then average over these snapshots to generate the electron density map. For the experimental map, we generate a $\{F_{obs}, \varphi_{obs}\}$ absolute scaled Fourier map. Once the electron density maps from MD simulations and experimental data are calculated, we compare the radial distribution (figure 2) of water around surface atoms. Calculating correlation coefficients to quantify the agreement between MD and experimental densities, we find that the experimental and MD-derived radial distribution functions correlate well, with Pearson correlation coefficients above 0.9. The various tested water models perform essentially identically.

Inferring protonation states

Unless the diffraction data has sub-atomic resolution (~1.0-0.7Å or better), it is not possible to directly detect hydrogen atoms from X-ray crystallography. Nevertheless, whether a proton is present or not affects the nearby solvent density. Running simulations with different protonation states and comparing the resulting water density with experiments help infer protonation states. For `1YTT`, for instance, in a histidine residue (`HIS116`, chain `A`), protonating `Nδ1` instead of `Nε2` results in clear peaks in the MD density that overlap with two crystal waters (figure 3). This effect is particularly remarkable because different prediction tools disagree: while Molprobity correctly assigns a proton to `Nδ1` only, `Gromacs` protonates `Nε2`. Note, however, that this strategy is only

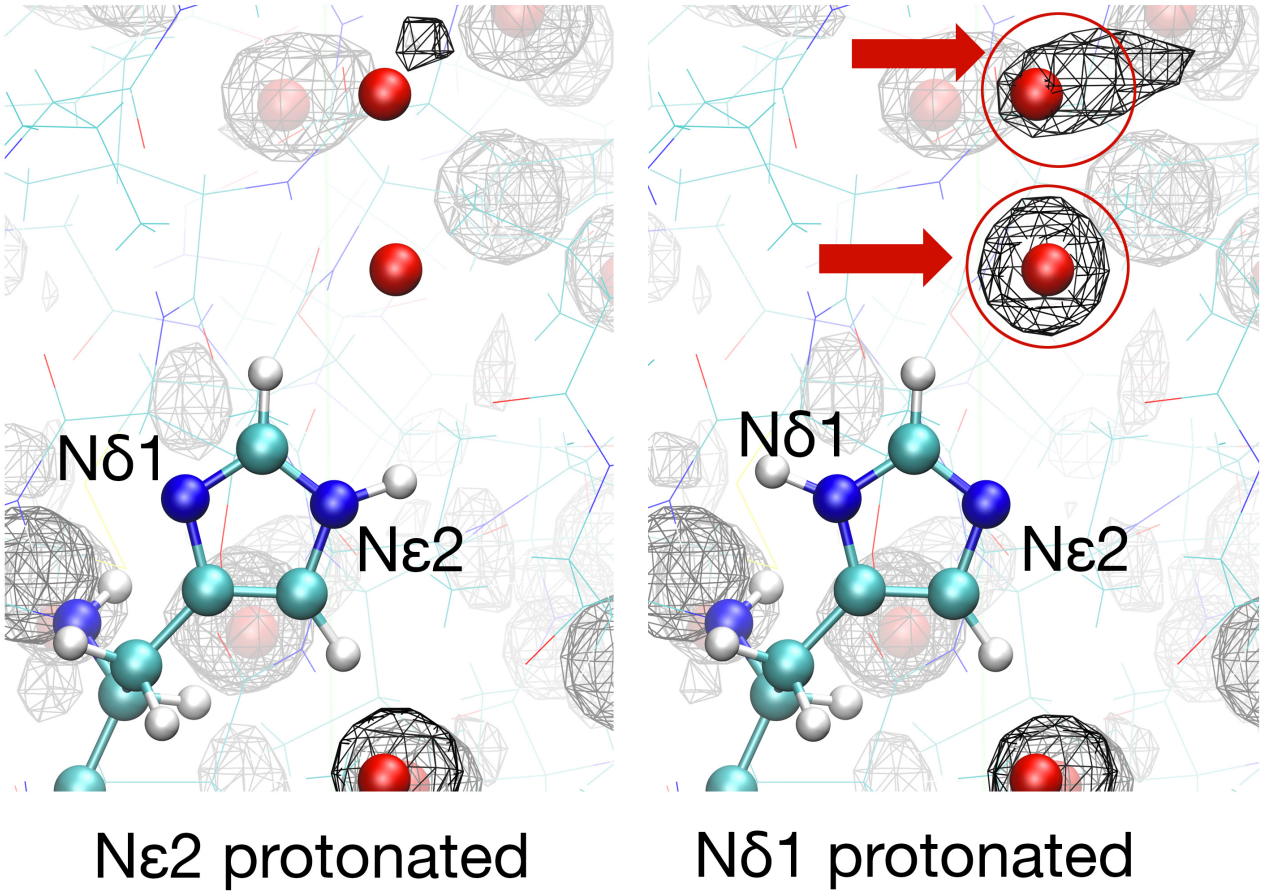


Figure 3: For a particular histidine (residue number `116`, chain `A`), the surrounding water density suggests that `Nε2` is not protonated. `Gromacs`, by analyzing the hydrogen bond network, protonates the wrong nitrogen, while `Molprobity` (Chen et al., 2010), which takes clashes into account, assigns it correctly. The maps shown here are calculated from MD snapshots in units of $e^-/Å^3$ (shown contoured at 0.6 $e^-/Å^3$).

applicable to atoms with sufficient solvent exposure.

Conclusion

While MD water models appear to perform well when averaged radially, they may not be sufficient for accurately reconstructing the real space solvent density on their own. Interestingly, we find that shortcomings in sampling methods and protein force fields are as much of a problem to accurately reconstructing water density as the MD water models themselves. We are nevertheless actively working on strategies for assigning protonation states and for developing hybrid schemes that utilize both experimental data and information from MD simulations that is complementary to the experimental data to describe solvation.

References

Adams, P. D., Afonine, P. V., Bunkóczi, G., Chen, V. B., Davis, I. W., Echols, N., J. J. Headd, L.-W. Hung, G. J. Kapral, R. W. Grosse-Kunstleve, A. J. McCoy, N. W. Moriarty, R. Oeffner, R. J. Read, D. C. Richardson, J. S. Richardson, T. C. Terwilliger and P. H. Zwart. (2010). *Acta Crystallographica Section D: Biological Crystallography*, **66(2)**, 213-221.

Berendsen, H. J., van der Spoel, D., & van Drunen, R. (1995). *Computer Physics Communications*, **91(1)**, 43-56.

Burling, F. T., Weis, W. I., Flaherty, K. M., & Brünger, A. T. (1996). *Science*, **271(5245)**, 72.

Chen, V. B., Bryan Arendall III, W., Headd, J. J., Keedy, D. A., Immormino, R. M., Kapral, G. J., Murray, L. W., Richardson, J. S., & Richardson, D. C. (2010). *Acta Crystallographica* **D66**: 12-21.

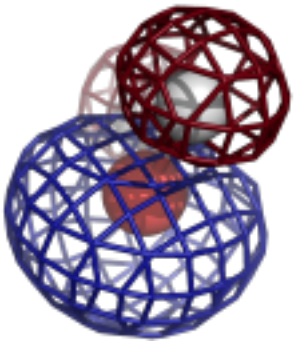
Holton, J. M., Classen, S., Frankel, K. A., & Tainer, J. A. (2014). *FEBS journal*, **281(18)**, 4046-4060.

Lounnas, V., Pettitt, B. M., & Phillips Jr, G. N. (1994). *Biophysical journal*, **66(3 Pt 1)**, 601.

Sonntag, Y., Musgaard, M., Olesen, C., Schiøtt, B., Møller, J. V., Nissen, P., & Thøgersen, L. (2011). *Nature communications*, **2**, 304.

Weichenberger, C. X., Afonine, P. V., Kantardjieff, K., & Rupp, B. (2015). *Acta Crystallographica Section D: Biological Crystallography*, **71(5)**, 1023-1038.

VOLUME EIGHT



COMPUTATIONAL CRYSTALLOGRAPHY NEWSLETTER

JANUARY MMXVII

O-O PAIRS, CRYO-EM DIFFERENCE MAP

Table of Contents

- Phenix News 1
- Crystallographic meetings 2
- Expert Advice
  - Fitting tips #13 – O-Pairs: The love-hate relationship of carboxyl oxygens 2
- Short Communications
  - multi_core_run(), yet another simple tool for embarrassingly parallel job 6
  - Phenix tool to compute a difference map for cryo-EM 8
- Article
  - Deploying *cctbx.xfel* in Cloud and High Performance Computing Environments 10

Editor

Nigel W. Moriarty, NWMoriarty@LBL.Gov

Phenix News

Announcements

Phenix Tutorial YouTube Channel launched

Video tutorials explaining how to run Phenix tools via the GUI are now available on the Phenix Tutorials YouTube channel. To access, search YouTube for "`phenix tutorials`" and look for the electron density icon in the header of this newsletter.

The videos give short introductions about the tool, summarize which input files are required, discuss the default values and explain how to run it and – when appropriate – cover a short summary of the results.

The following topics are covered:

- Running `phenix.refine`
- Calculating polder OMIT maps
- Running `phenix.real_space_refine`
- Using PDB tools
- Change `phenix.refine` parameters in the GUI

It is planned to add more videos in the near future. Feedback about the videos is very welcome (tutorials@phenix-online.org).

New programs

phenix.secondary_structure_validation

A command-line utility to quickly assess the quality of secondary structure annotations in the model file. The tool checks for presence of corresponding atoms in the model, evaluates number and length of hydrogen bonds and correctness of Ramachandran angles. It outputs individual secondary structure elements that contain errors as well as overall statistics. mmCIF and PDB formats are supported.

Phenix tool to compute a difference map for cryo-EM

See extended communication on pages 8–9.

Crystallographic meetings and workshops

Phenix Local Users Workshop, March 16, 2017

Location: Berkeley, CA. Users will attend seminars, hands-on tutorials and discuss specific issues relating to their research. Register by contacting the editor.

Macromolecular Crystallography School Madrid 2017 (MCS2017), May 5–10, 2017

Members of the *Phenix* team will be in attendance.

67th Annual Meeting of the American Crystallography Association, May 26–30, 2017

Location: New Orleans, LA. A *Phenix* workshop will be held on the 26th of May. See conference website for details.

24th Congress and General Assembly of the International Union of Crystallography, August 21–28, 2017

Location: Hyderabad, India. A *Phenix* workshop will be held on the 21st of August. See conference website for details.

Expert advice

Fitting Tip #13 - O-Pairs: The Love-Hate Relationship of Carboxyl Oxygens

Jane Richardson, Michael Prisant, Christopher Williams, Lindsay Deis, Lizbeth Videau and David Richardson, *Duke University*

One is generally taught that like charges repel each other, such as the negatively charged carboxyl groups of Asp or Glu sidechains. However, our studies found the O-to-O distance distribution for nearest-neighbor

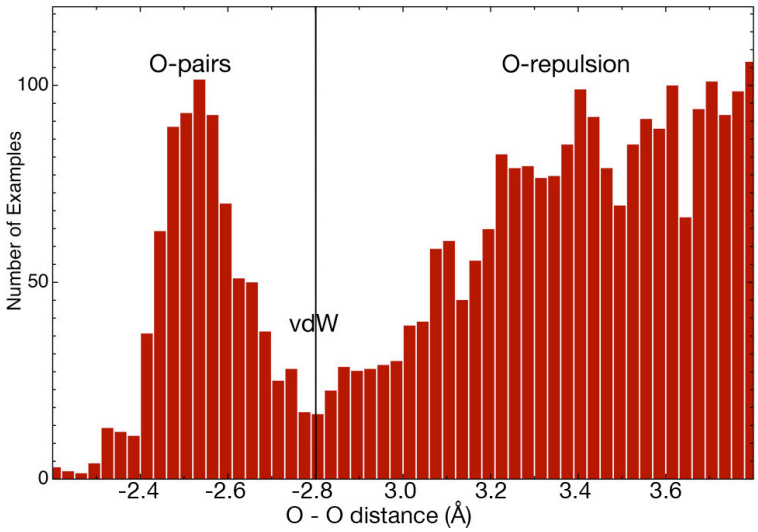


Figure 1: Distance distribution of nearest-neighbor sidechain carboxyl O atoms, measured relative to van der Waals contact (2.8Å O-O separation). From the Top8000 50%-homology SF <2.0Å dataset (Hintze 2016), residue-filtered only by electron density quality.

carboxyl oxygens to be strongly bimodal, as shown in Figure 1 (from the Top8000 50%-homology <2.0Å dataset, with residues filtered by electron-density quality). In a majority of cases the usual negative charges do indeed seem to repel one another, producing carboxyl O distances greater than the sum of their vdW radii (1.4 + 1.4 = 2.8Å), as seen on the right-hand side of the plot. However, a large number of clearly correct examples are about 0.3Å closer than van der Waals contact, producing the strong, well-separated peak at left in the distribution.

The low-barrier H-bond

These O-pairs represent a feature long known in small molecules (Emsley 1980), and called a "short", "strong", "charge-assisted", or "low-barrier" H-bond in the enzymatic literature (Cleland 1998; Gilli 2009; Ishikita 2013). There is indeed a hydrogen atom between the two carboxyl oxygens, although in protein x-ray structures its difference peak is very rarely visible even at sub-Å resolution (an H

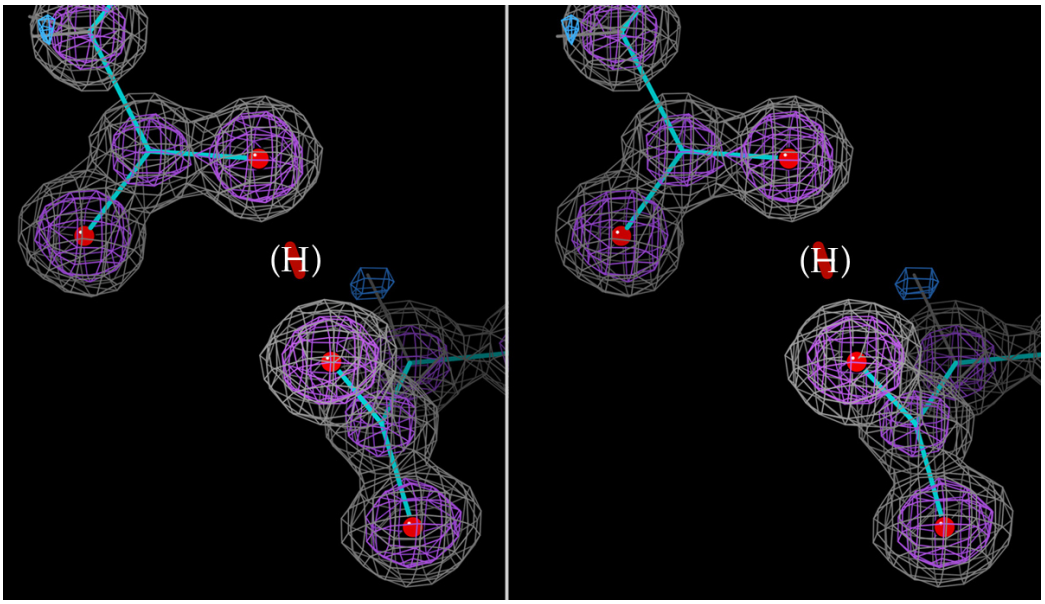


Figure 2: The short O-pair H-bond (2.48Å separation) between Asp233 and Asp246 of the 0.88Å-resolution 2p74 β lactamase (Chen 2007), in $2mF_o$-DF_c electron density. There must be a central H atom, but it is not observed above noise in the difference density (blue).

difference peak is seen for a His-Asp low-barrier H-bond in figure 3 of Kuhn 1998). Since protonation of a carboxyl is involved, O-pairs are somewhat more likely if the pH is low, either overall or for the local environment, such as for the four O-pairs seen in the 1oew endothiapepsin acid protease at 0.90Å (Erskine 2003), three of them at the surface and one buried. The classic no-barrier case is when the H is centered and bonded equivalently to both oxygens, at an O-O separation of 2.4 to 2.5Å. The example from the 2p74 β lactamase in figure 2 has a 2.48Å separation. Many small-molecule and protein examples are somewhat further separated, as documented in the distribution of figure 1, with the H atom presumably in equilibrium between two near-equivalent and closely-spaced energy wells. The NMR chemical-shift perturbation of the hydrogen (δ_H) is quite diagnostic, near 20-22 ppm if centered and 17-19 ppm if low-barrier but double-welled, as compared with 10-12 ppm for an "ordinary" asymmetric H-bond (Frey 2006).

O-pairs in enzyme catalysis

Short H-bonds are possible for many donors and acceptors if H-bond geometry and chemistry (e.g. pKa) are matched. They have been implicated as transition states in enzymatic mechanisms of many different types (reviewed in references above). However, catalytic O-pairs are usually transient and present only during a specific step in the reaction. Therefore the short interaction would not be seen in the resting state of the enzyme. Most examples in protein structures seem to be quite genuine but more or less accidental occurrences, without a preference for functional sites or strong conservation. They can be either buried or surface-exposed, and seem to have no strong preference among Asp-Asp, Glu-Glu, or Asp-Glu pairings. From the large literature understandably concerned with their important catalytic functions, one would not realize that short, low-barrier H-bonds are also found in the other regions of protein structures.

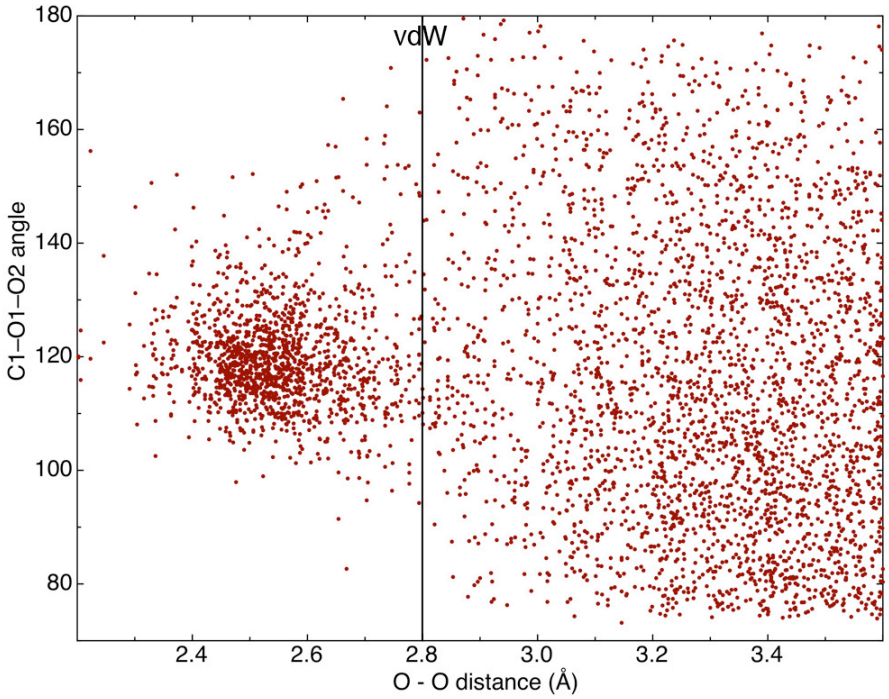


Figure 3: Distribution of the C2-O2-O1 angle as a function of O-O distance, showing a well-defined cluster below vdW contact and random above.

Geometric restrictions

The key to genuine occurrence of short O-pair carboxyl oxygens is satisfaction of a rather stringent set of restrictions on the geometry of their interaction (in addition to matched pKa, already true for Asp and Glu). A shared, approximately central H atom at short O-O distance is feasible only when the H-bond geometry is nearly ideal from both directions. In our quality-filtered reference data, these protein carboxyl short O-pairs only occur in a quite specific geometrical arrangement, where each O is in the plane of the other carboxyl and along the bond vector of the other's potential OH. Another way to describe this is that: a) the C1-O1-O2 and C2-O2-C1 angles are both 119° ±7° and b) the O1'-C1-O1-O2 and O2'-C2-O2-O1 dihedrals are (0°, 180°) ±20°, where 1 and 2 refer to the two Asp or Glu residues. Figure 3 shows the distribution of C2-O2-C1 angle versus O-O distance, clustered around 119° for short O-pairs. The dihedrals measuring coplanarity of each oxygen with the opposite carboxyl group show a similar distribution – well clustered for the short O-pairs and random at >vdW contact. The dihedral around the O1-O2 direction can take on any value except right at zero, but is often near 90° as in figure 2.

The bottom line

The take-home message for protein crystallography is that you are quite likely to encounter carboxyl O-pairs if you keep an eye out for them. Although not traditionally expected, they are reasonably frequent, more common at low than high pH, and quite justifiable if supported by a decent fit to clear density. The short O-O pair overlaps are almost never ≥0.4Å so they seldom trigger a MolProbity clash. If you see a 0.1-0.4Å small-overlap between carboxyl oxygens, it could just be a misfitting, but if it satisfies the correct geometry as described above then it is very likely to be genuine.

References

Chen Y, Bonnet R, Shoichet BK (2007) The acylation mechanism of CTX-M beta-lactamase at 0.88Å resolution, J Am Chem Soc 129: 5378-80

Cleland WW, Frey PA, Gerlt JA (1998) The low barrier hydrogen bond in enzymatic catalysis, *J Biol Chem* **273**: 25529-32

Erskine P, Coates L, Mall S, Gill RS, Wood SP, Myles DAA, Cooper JB (2003) Atomic resolution analysis of the catalytic site of an aspartic proteinase and an unexpected mode of binding by short peptides, *Protein Sci* **12**: 1741-9

Frey PA (2006) Isotope effects in the characterization of low barrier hydrogen bonds, in Isotope Effects in Chemistry and Biology (Kohen A, Limbach H-H, Eds) pp 975-993, CRC Press, Boca Raton FL

Gilli G, Gilli P (2009) The Nature of the Hydrogen Bond: Outline of a Comprehensive Hydrogen Bond Theory, Chapter 9, Oxford Scholarship Online, ISBN: 9780199558964

Emsley J (1980) Very strong hydrogen bonding, *Chem Soc Rev* **9**: 91-124

Hintze BJ, Lewis SM, Richardson JS, Richardson DC (2016) "MolProbity's ultimate rotamer-library distributions for model validation", *Proteins: Struc Func Bioinf* **84**,: 1177-1189

Ishikita H, Saito K (2013) Proton transfer reactions and hydrogen-bond networks in protein environments, *J Royal Soc Interface* **11**: 20130518

Kuhn P, Knapp M, Soltis M, Ganshaw G, Thoene M, Bott R (1998) The 0.78Å structure of a serine protease: *Bacillus lentis* subtilisin, *Biochemistry* **37**: 13446-52

multi_core_run(), yet another simple tool for embarrassingly parallel jobs

Robert D. Oeffner

Department of Haematology, University of Cambridge, Cambridge CB2 0XY, UK

`libtbx.easy_mp.multi_core_run()` is a new function in the *CCTBX* that allows the developer to run jobs in parallel. The function is a simplified version of a similar function `libtbx.easy_mp.parallel_map()` with some differences.

Brief introduction

For the past two decades it has been possible to implement the ability to execute on multiple CPU cores (multiprocessing) in most mainstream programming languages. But as the majority of PCs until the early 2000's were only single core machines there was little incentive for program developers to implement multiprocessing in their programs. Today, however, PCs can only be purchased with multiple CPU cores and a desktop PC with 16 or more cores is not uncommon. The expected behaviour of modern programs by users is naturally that the performance of the programs they run scale accordingly. The challenge for the developer is not only to identify which are the CPU intensive parts of the code, but also to parallelize the code in a manner and style that does not negatively impact on the maintainability of the code.

Foundation

Infrastructure has been implemented for transparent parallel job execution in the *CCTBX* (Echols, Bunkóczi, & Grosse-Kunstleve, 2013). These tools are versatile and present the developer with a common API that can be used for multithreading, multiprocessing as well as for remote job execution. The general nature of these functions means that they have several keyword arguments with default values. In `parallel_map()` only the `iterable` and the `func` keywords need to be specified.

multi_core_run()

A simpler version of `parallel_map()` called `multi_core_run(myfunction, argstuples, nproc)`, has now been added to the `libtbx.easy_mp` module. It uses the same infrastructure as `parallel_map()` but it has no default arguments and uses only python multiprocessing for parallelizing jobs. Like `parallel_map()` it expects the function arguments and the results of `myfunction()` to be pickleable.

The new function returns an iterator whereas `parallel_map()` returns a list of results. The items of the iterator are tuples corresponding to individual `myfunction()` jobs and structured like `(arg, res, err)`. Here `arg` is the tuple of arguments used for the job, `res` is the result of the job and `err` is anything printed to stderr by the job. Returning an iterator means the developer may process results from individual parallel jobs as they emerge. When the number of jobs to run is much larger than the number of CPU cores available this feature allows some display of progress or processing of intermediate results until the last item of the iterator emitted by `multi_core_run()` has been reached, i.e. when the last job has completed. Arguably this may also be achieved using `parallel_map()` by specifying a customised callback function as one of its many arguments. This callback function would then be called whenever an individual job completes. The design of `multi_core_run()` simplifies this so the developer need not provide an explicit callback function; progress is apparent as individual results emerge one by one from `multi_core_run()`.

`multi_core_run()` handles exceptions gracefully. Any exception message will be stored in the `err` string value of the item tuple

if a job crashes. Even severe exceptions crashing the underlying python interpreter of a job will get caught while remaining jobs continue to execute. This has been achieved by placing a `try-except` code guard inside the `for` loop that runs over the `libtbx.scheduling.parallel_for.iterator` when extracting the results of the individual jobs. In `parallel_map()` however, a `try-except` code guard is placed outside the `for` loop that runs over the `libtbx.scheduling.parallel_for.iterator`. Consequently if one job throws a segmentation fault in `parallel_map()` the exception handler will abort all outstanding calculations and raise a `Sorry` error. The design with `multi_core_run()` is to allow the developer to record any possible exceptions from individual jobs without terminating remaining jobs that run flawlessly. This is useful if the cause of the exception is merely the odd bogus input error that does not invalidate the other parallel calculations.

Example

In the following example a function in a separate module is defined as:

```
# testjob.py module

def RunMyJob( foo, bar ):
  import math
  return math.sqrt(foo)/bar
```

From a python prompt one can then issue the following commands

```
>>> import testjob
>>> from libtbx import easy_mp
>>>
>>> # define tuples of arguments for
RunMyJob()
>>> argstuples = [( 3, 4), (2, 3), (-7, 4)
]
>>>
>>> # execute RunMyJob() in parallel over 3
CPUs
>>> for args, res, errstr in
easy_mp.multi_core_run( testjob.RunMyJob,
argstuples, 3):
...   print "arguments: %s \nresult: %s
\nerror: %s\n" %(args, res, errstr)
...
```

This will print arguments, the corresponding results and any error string from `RunMyJob()`:

```
arguments: (3, 4)
result: 0.433012701892
error: None

arguments: (2, 3)
result: 0.471404520791
error: None

arguments: (-7, 4)
result: None
error: math domain error
Traceback (most recent call last):
  File "C:\Busers\oeffner\Phenix\dev-2063-
working\modules\cctbx_project\libtbx\schedu
ling\job_scheduler.py", line 64, in
 job_cycle
    value = target( *args, **kwargs )
  File "M:\SourceFiles\Python\testjob.py",
line 14, in RunMyJob
    return math.sqrt(foo)/bar

>>>
```

In the above example the calculation with the arguments `(-7, 4)` failed because it meant taking the square root of a negative number. A stack trace is stored in the error string that may be useful to the developer.

Conclusion

Multiprocessing is easy in principle but developers are often reluctant to exploit it due to lack of first hand coding experience. Taking a purist point of view most of the functionality of `multi_core_run()` is already duplicated in `parallel_map()` and thus redundant. However, with its somewhat different calling syntax and manner of returning values it may be more palatable for developers in cases where `parallel_map()` appears to be more cumbersome to use. The net result will then hopefully be that more and more code become parallelized regardless of the underlying tools used to achieve this.

References

Echols, N., Bunkóczi, G., & Grosse-Kunstleve, R. W. (2013). cctbx tools for transparent parallel job execution in Python. II. Convenience functions for the impatient. *Comput. Cryst. Newsl.* , *4*, 23–27.

Phenix tool to compute a difference map for cryo-EM

Pavel V. Afonine

Molecular Biophysics & Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA

While some high-resolution cryo-EM maps may be of exceptional quality, a typical cryo-EM map is still a low-resolution map (by crystallography standards, at least). Therefore locating and accurately placing ligands in such maps may not be a trivial task (figure 1). In crystallography a $\boldsymbol{\sigma}_A$ scaled mF_{obs}-DF_{calc} difference (or residual) map is the tool that is used routinely for locating yet unmodeled atoms. In cryo-EM there are no structure factors, observed F_{obs} or calculated F_{calc}, and therefore a difference map cannot be straightforwardly obtained. Naïvely, one could argue that it is possible to convert experimental cryo-EM map into structure factors by a Fourier transform. Similarly, it could be possible to calculate F_{calc} from the model using electron form-factors. This is a possibility, of course, but it is not without issues. The issues include:

a) Model completeness
b) B-factors inadequately refined
c) The bulk-solvent and scaling protocols designed for crystallography may not be appropriate
d) The map may contain artifacts left over from reconstruction that are away from the molecule and, if working in real space, pose no issues.

Any of these problems may be a showstopper if the map is calculated using structure factors. Typically most of these problems are present when working with cryo-EM data. Calculating a difference map in real space avoids problems all together and thus should to be a better choice. Furthermore, working with

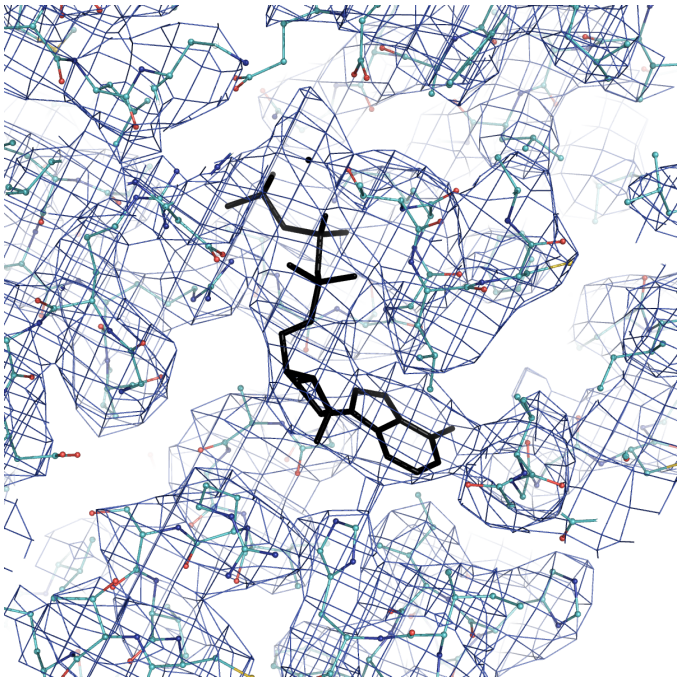


Figure 1: ATP in PDB model 5L4g superimposed on cryo-EM map emd_4002.

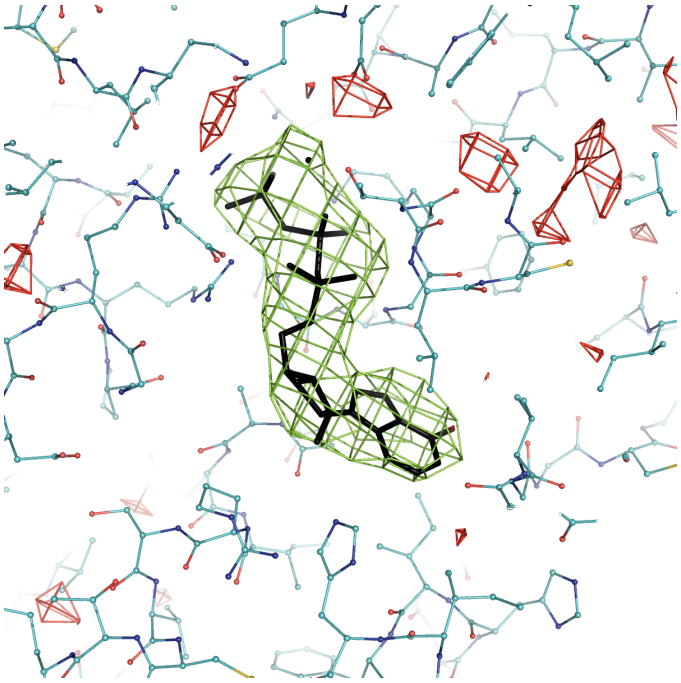


Figure 2: ATP in PDB model 5L4g in a difference density plot calculated using emd_4002.

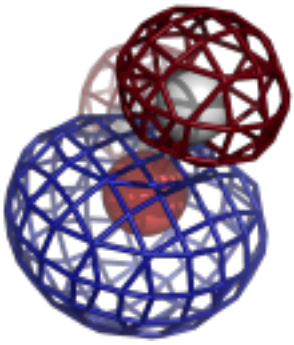
maps directly allows the use various local scaling methods that may enhance the utility of the resulting map.

We have implemented an option to calculate difference map in *Phenix* that is available starting nightly build `dev-2612`. Currently it is a command-line tool only, GUI will be available in future. Usage is as simple as

```
phenix.real_space_diff_map model.pdb map.ccp4 resolution=3.5
```

The program inputs are the model and map (actual map, not Fourier map coefficients!) to calculate a Fourier image of the model of specified resolution. This calculated map and the input (experimental) map are locally scaled and subtracted to produce the difference map. Figure 2 shows such a difference map for one of ATP molecules in 26S proteasome at 3.9Å resolution (Model, PDB code: 5L4g Map, EMDB code: emd_4002).

VOLUME EIGHT



COMPUTATIONAL CRYSTALLOGRAPHY NEWSLETTER

JULY MMXVII

1.12, SER & THR IN HELICES

Table of Contents

- Phenix News 20
- Crystallographic meetings 21
- Expert Advice
  - Fitting tips #14 – How Ser & Thr behave oddly on helices 21
- Short Communications
  - *phenix.mtriage*: a tool for analysis and validation of cryo-EM 3D reconstructions 25
  - More TLS validation with *phenix.tls_analysis* 26

Editor

Nigel W. Moriarty, NWMoriarty@LBL.Gov

Phenix News

Announcements

Phenix 1.12 release

The Phenix developers are pleased to announce that version 1.12 of Phenix is now available (build 1.12-2829). Binary installers for Linux, Mac OSX, and Windows platforms are available at the download site.[1] Highlights for this version include new tools for secondary structure validation, comparing multiple related structures, cryo-EM model and map validation, and calculation of real space difference maps. Video tutorials are now available on the new Phenix Tutorial YouTube channel.[2] A suite of tools for cryo-EM model building and structure refinement including automated map sharpening, segmentation and model building has also been added. Real space refinement now has an option to use a histogram-equalized map as a refinement target. Phaser version 2.8 has several new features, including unit cell refinement for EM data, more extensive use of eLLG in decision making and analysis, anomalous substructure determination (Phassade), improved tNCS detection and analysis, and improvements to SCEDS.

New programs

Structure Comparison

Tool for parallel validation and analysis of near-identical protein structures. Results include the analysis of ligands, rotamers,

[1] http://phenix-online.org/download/

[2] www.youtube.com/c/phenixtutorials

Ramachandran angles, missing atoms, water locations and B-factors.

phenix.mtriage

Command-line tool to validate cryo-EM models has been addded. GUI is available. Computes various map statistics including resolution estimates, FSC between half-maps, full and model maps. Outputs mask file used to compute FSC. Still under development.

phenix.real_space_diff_map

Command-line tool to compute difference map. This is a real-space analogue of Fo-Fc map mainly designed for cryo-EM maps.[1]

Crystallographic meetings and workshops

24th Congress and General Assembly of the International Union of Crystallography, August 21–28, 2017

Location: Hyderabad, India. A *Phenix* workshop will be held on the 21st of August. See conference website for details.

Expert advice

Fitting Tip #14 – How Ser & Thr behave oddly on helices

Jane Richardson and David Richardson

What is typical behavior on helices?

For most sidechain types, which have a Cγ group, the **p** (near +60°) rotamer of χ1 is completely forbidden on regular α-helix, because that group will seriously clash with the *i*-3 backbone CO of the preceding turn. At top left in figure 1, the clashes (red spike clusters) are shown between Cγ H atoms and the preceding helix turn (and a water), for a Thr misfit with the Cγ at **p** χ1. The no-**p**-on-helix rule was noted very early (McGregor 1987) and tabulated as percent occurrence for α, β, and 'other' categories in the penultimate rotamer library paper (Lovell 2000). In contrast, the **p** rotamer position is actually preferred for Ser and Thr on α-helix.

In general, **p** is by far the least preferred of the three χ1 angle optima (**p**, **m**, & **t**). For 14 of the 17 rotameric amino acids, the percentage of **p** averages only 10.9%, with a range of 6.2 to 17.7%. In contrast, for Ser and Thr the **p** rotamer is the commonest, at 48%. Leu is an exception in the other direction, at 0.8%, because its branched Cγ means that any χ2 value clashes with backbone if χ1 is **p**. This means that even before folding, Leu residues are set up with the right rotamers for α-helix, which is presumably a major reason that Leu has the highest helix propensity.

Serine and Threonine

Of course, the answer to this conundrum is that for Ser or Thr, the Oγ can make a short, favorable H-bond with the *i*-3 CO, as seen at top right in figure 1. When the misfit Thr is rebuilt, every criterion gets better, as listed in figure 1. At this high resolution it is easy to see what happens. In order to fit the reversed O and C branches into their strong density peaks, the Cβ is forced entirely out of its density and has huge geometry distortions. The right answer of an H-bond seems really obvious if you have all-atom contacts to judge by, which was not the case for these depositors in the 1990s. Also, one is used to expecting an OH with solvent exposure to point outward, not inward.

This tucked-in **p** rotamer arrangement results in a bifurcated H-bond for the *i*-3 CO: one branch is from the Thr Oγ and the other is the normal helical H-bond. We do not know of a study addressing this specific issue, but our

[1]See http://phenix-online.org/newsletter/CCN_2017_01.pdf for details.

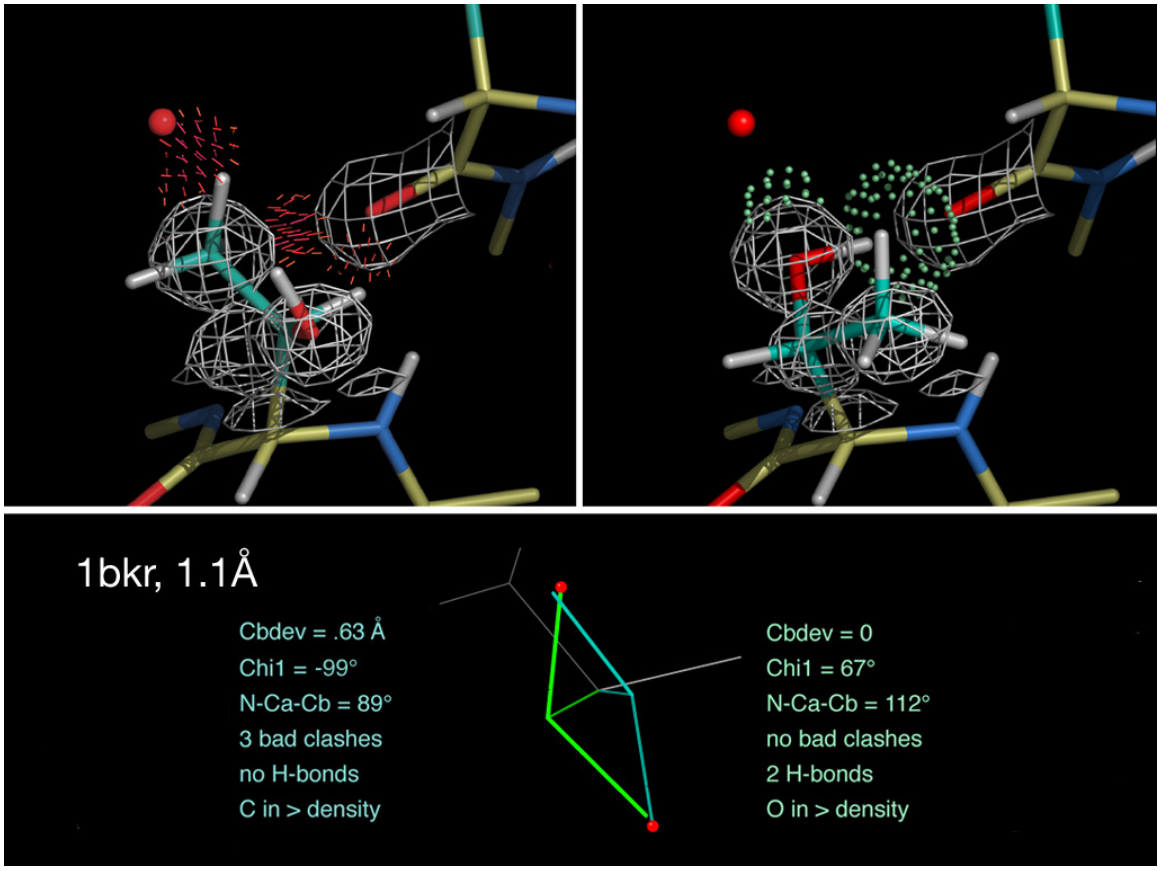


Figure 1: A misfit (left) vs corrected (right) Thr sidechain on an α-helix at 1.1Å resolution. The methyl group clashes with the *i*-3 carbonyl O and also with a nearby water (red ball), while in contrast the OH in that position can make two good H-bonds. Geometry and fit to density improve greatly, as listed in the lower panel. All-atom clashes are shown as clusters of red spikes, and H-bonds as lenses of pale-green dots. Thr 101 in 1bkr (Banuelos 1998).

impression is that such bifurcated H-bonds are in general somewhat weaker than a good single H-bond. That idea is supported by the fact that Ser and Thr have quite low helix propensities. Asn, also disfavored in helix, can make a similar bifurcated H-bond from its sidechain amide to the *i*-4 CO, in an **m-80** rotamer; however, more often the amide lies against the outer surface of the *i*-4 peptide, in rotamer **m-20** (Lovell 1999)

In membrane proteins

The H-bonded **p** conformation of Ser and Thr on helix is fairly common in globular proteins. It is even more favorable on trans-membrane helices, where there is very seldom a polar group to H-bond with an outward-pointing OH. A membrane-buried helical Ser or Thr is therefore very likely to adopt a **p** rotamer, where presumably it gains more from forming an H-bond than the helix loses from the bifurcation destabilizing a backbone H-bond somewhat. Figure 2 illustrates one of the many Ser/Thr H-bonded to their trans-membrane helices in the 5u9w trimeric transporter structure (Hirschi 2017). The left panel shows chain A Ser 110 within the membrane trimer. The right panel is a closeup, showing the inward-pointing

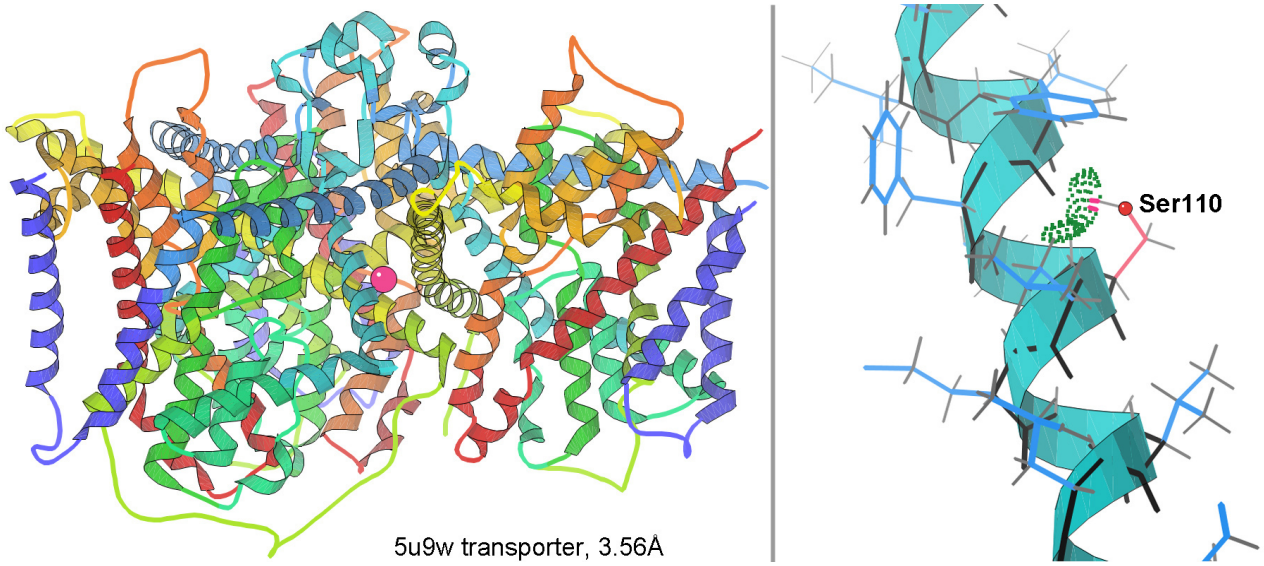


Figure 2: A **p**-rotamer, *i*-3 H-bonded serine on a trans-membrane helix at 3.56Å. Left panel shows ribbons for the trimeric transporter molecule viewed in the plane of the membrane, with Ser 110 as a hot-pink ball. Right panel is a closeup of Ser 110 on its helix, with green dot lenses for the bifurcated H-bond, one H-bond branch from the *i* -3 CO to the Ser sidechain OH and the other branch the normal helical H-bond. From 5u9w (Hirschi 2017)

sidechain, the bifurcated H-bond, and the regular α-helix.

The bottom line

In any protein structure, it is worth considering an i-3 H-bonded **p** rotamer for a Ser or Thr on an α-helix. For a trans-membrane helix, that arrangement is the dominant one, unless the sidechain happens to be next to a polar group or solvent channel. Therefore, if the Ser/Thr orientation is unclear in a membrane protein by low-resolution X-ray or high-resolution cryoEM, choose the default H-bonded **p** rotamer.

References:

Banuelos S, Saraste M, Carugo KD (1998) Structural comparisons of calponin homology domains: impications for actin binding, *Structure* **6**: 1419-1431

Hintze BJ, Lewis SM, Richardson JS, Richardson DC (2016) MolProbity's ultimate rotamer-library distributions for model validation, *Proteins: Struct Func Bioinf* **84**: 1177-1189

Hirschi M, Johnson ZL, Lee S-Y (2017) Visualizing multistep elevator-like transitions of a nucleoside transporter, *Nature* **545**: 66-70

Lovell SC, Word JM, Richardson JS, Richardson DC (1999) Asparagine and glutamine rotamers: B-factor cutoff and correction of amide flips yield distinct clustering, *Proc Natl Acad Sci USA* **96**: 400-405

Lovell SC, Word JM, Richardson JS, Richardson DC (2000) The Penultimate Rotamer Library, *Proteins: Struct Func Genet* **40**: 389-408

McGregor MJ, Islam SA, Sternberg MJE (1987) Analysis of the relationship between side-chain conformation and secondary structure in globular proteins, *J Mol Biol* **198**: 295-310

FAQ

What is in the MTZ output by *phenix.refine*?

The MTZ file from *phenix.refine* contains four kinds of data:

1. Copy of input data. Why? For convenience and consistency. Inputs may not necessarily be in MTZ format and may be spread across multiple files of different format.
2. Data that were actually used in refinement. Why? A user has options to cut resolution from both ends, as well as apply cutting by sigma. Plus, *phenix.refine* may choose not to use a handful of reflections as outliers (Read, 1999). So it may be good to have set of reflections that were used in given refinement run.
3. Model in reciprocal space (F_{model}). Why? This is a reciprocal space representation of what's in model file except that it is richer because contains not only atomic model (F_{calc}) but also solvent contributions (bulk and non-uniform) as well as all scales. F_{model} taken from this array and F_{obs} described in item 2 are expected to reproduce reported R_{factor} exactly.
4. Fourier map coefficients ($2mF_{obs}$-DF_{model}, mF_{obs}-DF_{model}, anomalous map if applicable).

Item 1 and 3 can be optional because with trivial scripting one can obtain F_{model} using data from item 2 and 4.

Item 1 does duplicate data and it's not unreasonable to assume they are still available by the time you finalize your structure. However, space is cheap and it is much easier to have relevant arrays of data centralized in one place (file) rather than scattered across hard drive.

References:

Read, R. J. 1999. "Detecting Outliers in Non-Redundant Diffraction Data." *Acta Crystallographica Section D: Biological Crystallography* 55 (10): 1759–64. doi:10.1107/S0907444999008471.

phenix.mtriage: a tool for analysis and validation of cryo-EM 3D reconstructions

Pavel V. Afonine

Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

phenix.mtriage is a new *Phenix* tool for analysis of cryo-EM derived maps. Given a map it provides a basic statistics about the map, such as box parameters (three-dimensional gridding), map origin, min/max/mean map values, histogram of map values, etc. Also, it estimates the map resolution obtained by map perturbation and that we refer to as d_{99}. In a nutshell, this measure indicates up to what resolution Fourier map coefficients corresponding to the real-space map can be omitted before the map starts changing significantly. This is done by Fourier transforming the map to obtain a full set of possible Fourier map coefficients. Then highest resolution terms are removed gradually and for each chunk of removed terms the correlation between the original map and the map corresponding to the truncated set of Fourier coefficients is computed. Obviously, at the start the correlation is 1 and it decreases with the amount of removed terms. Empirically we find that 0.01 in correlation drop is indicative of a significant change in the map, so we note the resolution cut-off at which this occurs, d_{99}.

Providing half-maps will enable more validation: the FSC plot will be calculated (and available as a graphic picture) and the resolution derived at FSC=0.143 ("gold standard") will be shown (d_{FSC}). Additionally, three map histograms (full and two for each half-map) will be shown. The all three histograms are expected to be similar.

Finally, if atomic model is provided then even more statistics will be generated. It is possible to use an atomic model to estimate map resolution by generating a series of model-calculated maps at different resolutions and comparing them with the experimental map. The resolution of the model-calculated map (d_{model}) that maximizes the correlation is considered as the most representative of the map resolution. Overall B-factor is calculated and reported as part of d_{model} calculation. Also, it is possible to calculate the correlation between model and experimental maps in Fourier space, similarly to FSC for half-maps. It is informative as it gives an estimate up to what resolution (d_{FSC_model}) there is at least some signal (FSC=0). Clearly, these measures are model-dependent and as accurate as accurate the model is.

As will be demonstrated elsewhere, it is informative to consider all four resolution estimates d_{FSC}, d_{99}, d_{FSC_model}, d_{model} and overall B as collectively they may indicate issues ranging from as trivial as typos to particular quirks of the data.

This is an ongoing development and details will be published elsewhere. More functionality is likely to appear in future. *phenix.mtriage* is available as a command line tool and in *Phenix* GUI. Users are encouraged to provide their feedback and requests for options.

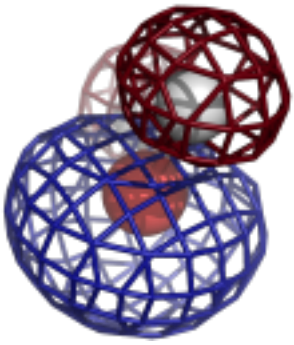
More TLS validation with *phenix.tls_analysis*

Pavel V. Afonine

Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

phenix.tls_analysis is a *Phenix* tool for analysis and validation of refined TLS parameters (Computational Crystallography Newsletter (2015). Volume 6, Part 2., page 27). The original implementation validates TLS matrices by interpreting them in terms of parameters of elemental motions. This approach is efficient at catching cases when refined elements of T, L and S matrices cannot describe a motion in turn indicating that these matrices are invalid. The TLS formalism assumes that atomic motions being modeled are harmonic. However, this assumption is not enforced neither in refinement protocols nor in decomposition of TLS matrices into parameters of translation-vibration motion. This makes it possible that refined TLS parameters do describe a motion but the kind of motion they describe is beyond the scope of TLS theory. The proposed way to verify that TLS parameters describe a harmonic motion consists of following steps. First, individual atomic displacement (ADP) parameters are calculated from TLS matrices using the known formula $U_{TLS} = T + ALA^t + AS + S^tA$. Then refined TLS matrices are used to extract parameters of concerted motions that they describe. These parameters include amplitudes of translations and vibrations, positions and orientations of corresponding axes, etc. In turn, these parameters are used to generate an ensemble of explicit atomic models with each model from the ensemble being consistent with TLS matrices. Having the ensemble of models allows converting positional uncertainty of each atom into corresponding displacement parameter, $U_{ensemble}$. Since $U_{ensemble}$ and U_{TLS} originate from the same source they are expected to match. Their mismatch indicates that motions that corresponding TLS matrices parameterize are anharmonic and, consequently, TLS matrices themselves are invalid. This analysis is now part of *phenix.tls_analysis* tool; details will be described elsewhere.

VOLUME NINE



COMPUTATIONAL CRYSTALLOGRAPHY NEWSLETTER

JANUARY MMXVIII

1.13, CRYO-EM, C BETA, GEMMI

Table of Contents

- Phenix News 1
- Crystallographic meetings 2
- Expert Advice
  - Fitting tips #15 – New help to make your 2.5–4Å cryoEM structure even better 2
- Short Communications
  - Tools for model-building with cryo-EM maps 7
  - Gemmi – a new MX library 13
  - Overfitting to Ramachandran and geometry criteria in the cryoEM Model Challenge 16
- Articles
  - Cβ deviations and other aspects in Amber versus CDL refinements 21
  - A few benchmark tests of various compilers on Linux and Windows 25

Editor

Nigel W. Moriarty, NWMoriarty@LBL.Gov

Phenix News

Announcements

Phenix 1.13 release

The Phenix developers are pleased to announce that version 1.13 of Phenix is now available (build 1.13-2998). Binary installers for Linux, Mac OSX, and Windows platforms are available at the download site.[1] Highlights for this version include new tools and feature enhancements:

- phenix.map_to_model - better symmetry support and improved runtime efficiency
- phenix.structure_search - structural library and internal support for mmCIF
- phenix.ligand_identification - limit ligand size for search
- Phaser 2.8.1 - various bug fixes
- Structure Comparison - more/improved validation information (ligands, waters, cis/trans peptides, HIS protonation)
- GUI - automatic validation after phenix.real_space_refine; visual improvements in validation
- Internal bug fixes and performance improvements

New video tutorials have been added to the Phenix Tutorial YouTube channel.[2] There include and overview of the cryo-EM tools in Phenix, a step-by-step guide on how to run MolProbity (web interface and Phenix GUI) and MolProbity: All atom contacts tutorial.

[1] http://phenix-online.org/download/

[2] www.youtube.com/c/phenixtutorials

Crystallographic meetings and workshops

The Astbury Conversation, Understanding Life in molecular detail, April 16–17, 2018

Location: University of Leeds. See website http://www.astburyconversation.leeds.ac.uk for details.

Expert advice

Fitting Tip #15 – New help to make your 2.5–4Å cryoEM structure even better

Christopher Williams, Lizbeth Videau, and Jane Richardson
Duke University

We join cryoEM structural biologists in being very excited and interested in the new, unprecedentedly higher-resolution structures now possible, and are working to develop new model-building, assessment and improvement tools suitably tuned for these new needs. We acted as assessors for the EMDB's CryoEM Model Challenge to try out the relevance and usefulness of such tools. This tip briefly summarizes progress so far, both within Phenix and MolProbity and our favorites from elsewhere.

In the 2.5 to 4Å resolution range the model-to-map fit can be quite convincing, but it often has enough leeway to prevent a unique answer. We find that refinement is therefore able to satisfy the traditional validation criteria of geometry, map fit, Ramachandran, and rotamers and still be stuck in the wrong local-minimum conformation in many places. Therefore, new criteria are needed that are not yet used in refinement and that integrate measures somewhat more broadly. We feel the two most generally useful such new tools for cryoEM models of protein are the CaBLAM backbone analysis from our lab (see below) and the EMRinger tool from the Fraser lab (Barad 2015). EMRinger looks for satisfaction of the expected chi1 angles at the Cβ atom – not to analyze rotamers, but to check for problems in the backbone that turn the Cα-Cβ in the wrong direction. You can get this analysis on your own structure at emringer.com. Tools for nucleic acid models are discussed below.

CaBLAM

The CaBLAM software from the Duke Phenix developer team (Williams 2013; 2018) uses Cα virtual dihedrals to follow the backbone intent of the relatively low-resolution model and density, and uses the virtual dihedral between adjacent peptide CO orientations to diagnose problems with the detailed conformation that has been fitted. A cryoEM example from KiNG 3D graphics

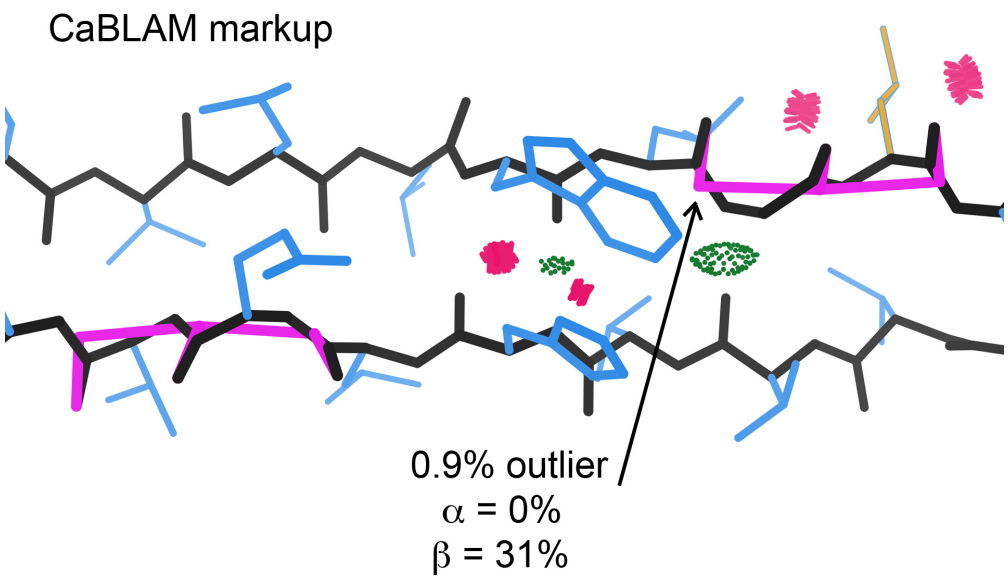


Figure 1: CaBLAM markup for incorrect peptide orientation on a CryoEM Model Challenge β-hairpin.

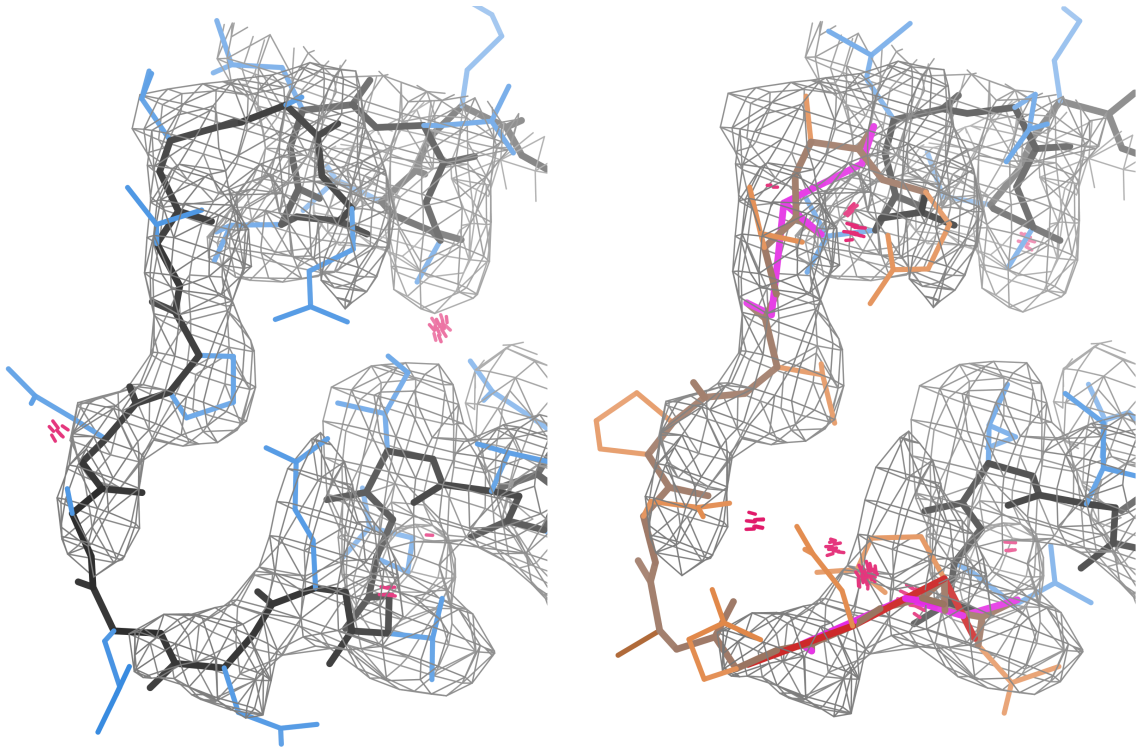


Figure 2: CaBLAM often flags one or both ends of a local sequence misalignment, even when no other validation metrics do. This figure compares the CryoEM Model Challenge target 1 cryoEM structure 4udv at 3.3Å (left) with the submitted model T0001EM188_1 (right, with misaligned region in brown & peach).

in MolProbity is shown in figure 1, where CaBLAM 1% outliers flag two places where 3 successive CO groups have been pointed in the same direction rather than alternating. This is never seen in good reference data and prevents some of the β-type H-bonds. As shown, when clicked on, the CaBLAM markup also reports on probability that the local structure is helix (0% here) or β-strand (31% here). This information can also be accessed in Phenix from the command-line.

The CaBLAM algorithm can diagnose many different sorts of problems, from wildly improbable peptide orientations within helices or beta strands, to analysis of Cα-only models, to sidechain-mainchain switches, to local sequence misalignments such as the example shown in figure 2. The central region of a local sequence misalignment often shows poor map fit, rotamers, and sterics at higher resolutions, but is not clearly diagnostic in the 2.5–4Å range. Even the ends can often be fit to avoid Ramachandran outliers, but are most reliably flagged by clashes and CaBLAM outliers. Note, though, that such outliers mean there is some fairly severe problem, but it is only sometimes a sequence misalignment.

Cis-nonPro and twisted peptides

CaBLAM also often flags *trans* residues that should have been fit as *cis* or vice versa. *Cis* prolines are fairly common at 5%, but *cis*-nonPro peptides are extremely rare at ~1 in 3000 or 0.03% (Williams 2015). Over the last decade, *cis*-nonPro were hugely overused without anyone noticing until recently; they are in the libraries and prior probabilities are not considered yet in model-building (the Phenix teams are working on that). There use is even more frequent than random, because a *cis* peptide seems to fit better into a shortened, curled-in loop. Now that MolProbity, Phenix, and Coot all flag *cis*-nonPro (Williams 2018), their overuse has decreased dramatically in crystal structures. CryoEM structures need also to watch out for this problem. Figure 3 shows that at 2.2Å the map fit can actually tell the difference of *cis* vs *trans* if you look for it. At ≥3Å the map

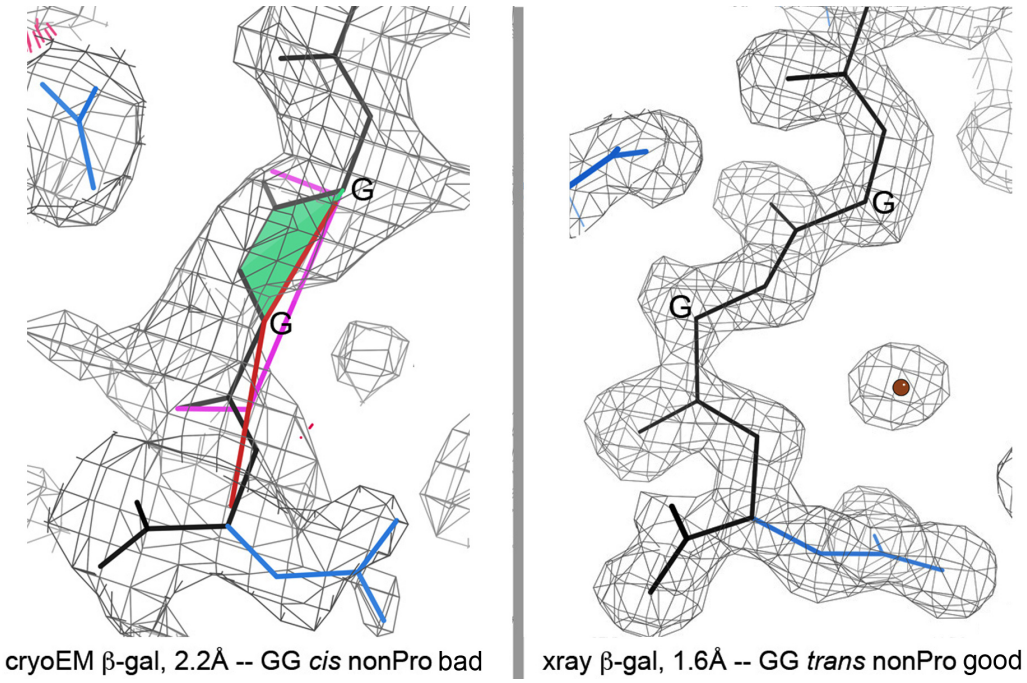


Figure 3: CaBLAM graphical flagging of an incorrect Gly-Gly *cis*-nonPro in the cryoEM 2.2A 5a1a β-galactosidase (left), compared with a clear fit as *trans* from the 1.6Å xray 4ttg (right; Wheatley 2015). At 2.2Å, the 5a1a cryoEM map, as well as CaBLAM, actually suggests that this local fit is incorrect.

cannot possibly distinguish, so you should only fit *cis* if it is known to occur for your protein at higher resolution (for instance, for 3 *cis*-nonPro at functional sites in the carbohydrate-active β-galactosidase).

RNA: ribose puckers and backbone suite conformers

The phosphates in nucleic acid structures are negatively charged and have lowered density in cryoEM maps, just as protein sidechain carboxyl groups. As shown in figure 4 below, the more positively-charged bases, in contrast, are stronger in cryoEM than in x-ray maps. Base-pairs are thus a really excellent way to locate double helices in both RNA and DNA.

RNA backbone is a tube between ribose lumps, very hard to fit in detail for either x-ray or cryoEM. Fortunately, MolProbity has developed validation that can infer ribose pucker (3'-endo or 2'-endo) with high reliability just from the well-seen and interpreted PO_4 and base positions (Richardson 2008; Jain 2014). That "P-perp" criterion enables pucker-specific targets in Phenix refinement, which can correct some of the problems, and it is easily seen by eye, to aid manual rebuilding for harder cases.

RNA backbone has distinct conformers when analyzed as sugar-to-sugar "suites", of which 54 have been identified and named by community consensus (Richardson 2008). These conformers can be used either for model-building or validation, and are analyzed in MolProbity. [Note that the wwPDB uses a criterion of "suiteness" (analogous to "rotamericity"), which is much less powerful or useful than puckers and conformers, since it is greatly affected by the percentage of A-form double helix in the structure.]

DNA conformation is simpler overall than RNA, since it is nearly always close to B-form double helix. However, it is actually more difficult than RNA to validate, because it is locally more flexible, so less restrained and therefore also more subject to error. Standard programs such as 3DNA (x3dna.org; Colisanti 2013) are helpful, but refinement may be using them by now. A new tool for DNA backbone conformers, to which no one is

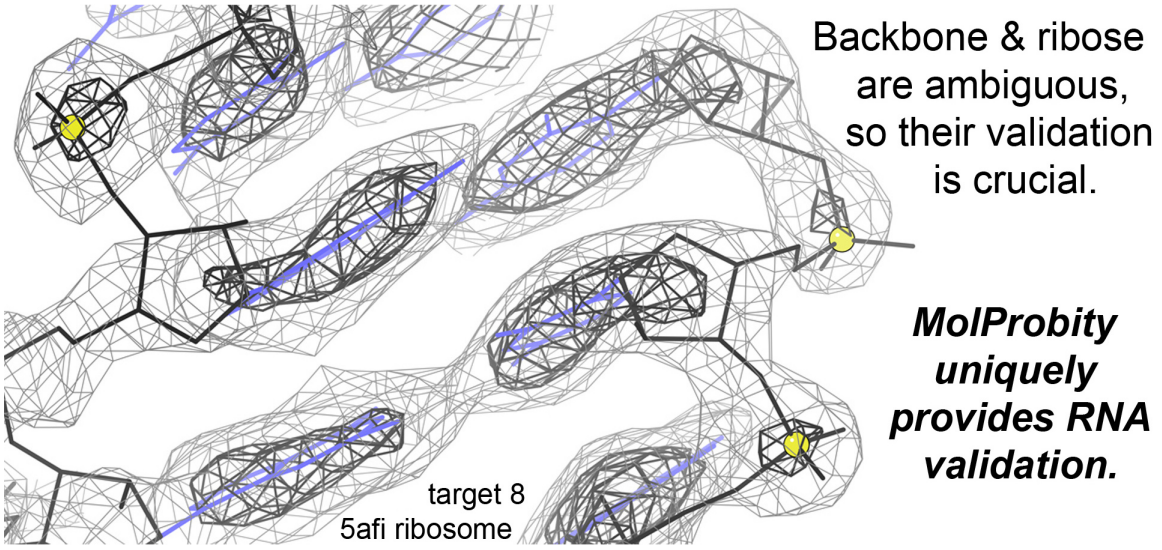


Figure 4: RNA structure seen in the cryoEM map of a ribosome at 2.9Å (5afi; Fischer 2015). Positive bases are very strong, while negative phosphates are weaker than in crystallographic density.

yet refining, is available at dnatco.org (Schneider 2018). Their data is not quality-filtered, so some of the conformers are not actually possible, but most are good and its use could help improve accuracy considerably.

The bottom line

Even if your cryoEM model has been fit and refined so as to get rid of nearly all outliers on the traditional validation metrics (geometry, Ramachandran, rotamer, and sterics), at 2.5 to 4Å resolution there are very likely to still be local regions stuck in the wrong local minimum. However, new tools are being developed which are both independent and also sensitive enough to diagnose locally misfit regions and guide their correction.

References

Barad BA, Echols N, Wang R Y-R, Cheng YC, DiMaio F, Adams PD, Fraser JS (2015) "EMRinger: Sidechain-directed model and map validation for 3D electron cryomicroscopy", *Nature Methods* **12**: 943-946

Bartesaghi A, Merk A, Bannerjee S, Matthies S, Wu X, Milne J, Subramaniam S (2015)"2.2 A resolution cryo-EM structure of beta-galactosidase in complex with a cell-permeant inhibitor" Science **348**: 1147-1151 [5a1a]

Colasanti A, Lu X-J, Olson WK (2013) "Analyzing and building nucleic acid structures with 3DNA", *JoVE*, **74**, e4401

Fischer N, Neumann P, Konevega AL, Bock LV, Ficner R, Rodnina MV, Stark H (2015) "2.9Å structure of *E coli* ribosome-EF-Tu complex by cs-corrected cryo-EM", *Nature* **520**: 567-570

Fromm SA, Bharat TAM, Jakobi AJ, Hagen WJH, Sachse C (2015) "Seeing tobacco mosaic virus through direct electron detectors" *J.Struct Biol* **189**: 87-97 [4udv]

Jain S, Kapral G, Richardson D, Richardson J (2014) "Fitting Tips #7: Getting the pucker correct in RNA structures", *Comput Crystallogr Newsletter* **5**: 4-7

Li N, Zhai Y, Zhang Y, Li W, Yang M, Lei J, Tye BK, Gao N (2015) "Structure of the eukaryotic MCM complex at 3.8Å", Nature 524: 186-191 [3ja8]

Richardson JS, Schneider B, Murray LW, Kapral GJ, Immormino RM, Headd JJ, Richardson DC, Ham D, Hershkovits E, Williams LD, Keating KS, Pyle AM, Micallef D, Westbrook J, Berman HM (2008) "RNA

Backbone: Consensus all-angle conformers and modular string nomenclature." *RNA* **14**: 465-481
Schneider B, Bozikova P, Necasova I, Cech P, Svozil D, Cerny J (2018) "A DNA structural alphabet provides new insight into DNA flexibility", *Acta Crystallographica* **D 74**:52-64
Wheatley RW, Juers DH, Lev BB, Huber RE, Noskov SY (2015) "Beta-galactosidase (E. coli) in the presence of potassium chloride" *Phys Chem Chem Phys* **17**: 10899-10909 [4ttg]
Williams CJ, Hintze BJ, Richardson JS, Richardson DC (2013) "CaBLAM: Identification and scoring of disguised secondary structure at low resolution", *Computational Crystallography Newsletter* **4**: 33-35
Williams CJ, Richardson JS (2015) "Fitting Tips #9: Avoid excess *cis* peptides at low resolution or high B", *Comp Cryst Newsletter* **6**: 2-6
Williams CJ, Hintze BJ, Headd JJ, Moriarty NW, Chen VB, Jain S, Prisant MG Lewis SM, Videau LL, Keedy DA, Deis LN, Arendall WB III, Verma V, Snoeyink JS, Adams PD, Lovell SC, Richardson JS, Richardson DC (2018) MolProbity: More and better reference data for improved all-atom structure validation, *Protein Science* **27**: 293–315

FAQ

What happens when the Phenix GUI prompts me to send an error report?

If the Phenix GUI pops up a window to send an error report, it is an opportunity to work with a Phenix developer and solve the current situation that caused the problem. There are a few things to do that can make this a useful and efficient experience for all.

1. Type in the correct email address. A Phenix developer answers all emails that contain a unique issue and user combination. Without a functioning email address, the process is cut short.
2. Consider typing in a description. This will help us diagnose the problem and may also help you with the standard questions from a developer that usually include:
   a. What OS are you using?
   b. What programs and options were you using when the problem occurred?
   c. Can you provide the inputs?
3. If you run across the same problem, you can rest assured that the problem has been registered and there is no need to resend the same issue.
4. Consider checking the website, phenix-online.org, for later versions of Phenix. Many issues are resolved daily by the developers and installing the latest Phenix may solve your problem. You can have multiple installations and choice to use any of those installed on your computer.

These tips can help with solving the problems which a high priority of the developers. Providing the inputs and a description of the events that led up to the crash is critical to reproduction of the crash by a developer and the eventual resolution.

Tools for model-building with cryo-EM maps

Tom Terwilliger

Los Alamos National Laboratory, Los Alamos NM 87545

New Mexico Consortium, 100 Entrada Dr, Los Alamos, NM 87544

Introduction

There are new tools available to you in *Phenix* for interpreting cryo-EM maps. You can automatically sharpen (or blur) a map with `phenix.auto_sharpen` and you can segment a map with `phenix.segment_and_split_map`. If you have overlapping partial models for a map, you can merge them with `phenix.combine_models`. If you have a protein-RNA complex and protein chains have been accidentally built in the RNA region, you can try to remove them with `phenix.remove_poor_fragments`. You can put these together and automatically sharpen, segment and build a map with `phenix.map_to_model`.

Sharpening a map with phenix.auto_sharpen

The `phenix.auto_sharpen` tool is designed to find the optimal sharpening for a cryo-EM (or crystallographic) map based on maximizing the detail in the map as well as the connectivity. Sharpening of the map is carried out by applying an overall sharpening B-factor to the amplitudes of the Fourier transform of the map up to the nominal resolution of the map. Beyond that resolution the amplitudes are damped. In this method, the key parameter is the value of the sharpening B-factor.

The detail in the map is represented by the surface area of contours at a fixed contour level. The (lack of) connectivity is represented by the number of regions in the map at that contour level; the contour level is chosen to enclose about 20% of the volume of the molecule. The target for optimization is the adjusted surface area calculated as the surface area minus a scale factor times the number of regions in the map. The scale factor is set automatically by requiring the adjusted surface area to be equal at the lowest and highest values of the sharpening B-value tested. The sharpening B-value is then adjusted to maximize the adjusted surface area. The auto-sharpen procedure creates a new map that is in the same position and has the same grid as the original map.

You can apply the auto-sharpening procedure globally (the same sharpening B-factor is applied to the entire map) or locally (a different sharpening B-factor is applied to each part of the map). In practice, typically the global sharpening is just as good as local sharpening. The local sharpening procedure uses the segmentation procedure in `phenix.segment_and_split_map` to find the regions in the map where the molecule is located. For each such region, a box of density is cut out surrounding the region and a local sharpening B-value is identified and applied to the grid points in the box. The density for points that are in more than one box is their average, weighted based on the distance of each point from the center of corresponding boxes. Points outside any of these boxes are sharpened with the global sharpening B-value.

You can also refine not just the overall sharpening B-factor, but also the high-resolution cutoff where the sharpening turns into blurring and the sharpness of the transition from sharpening

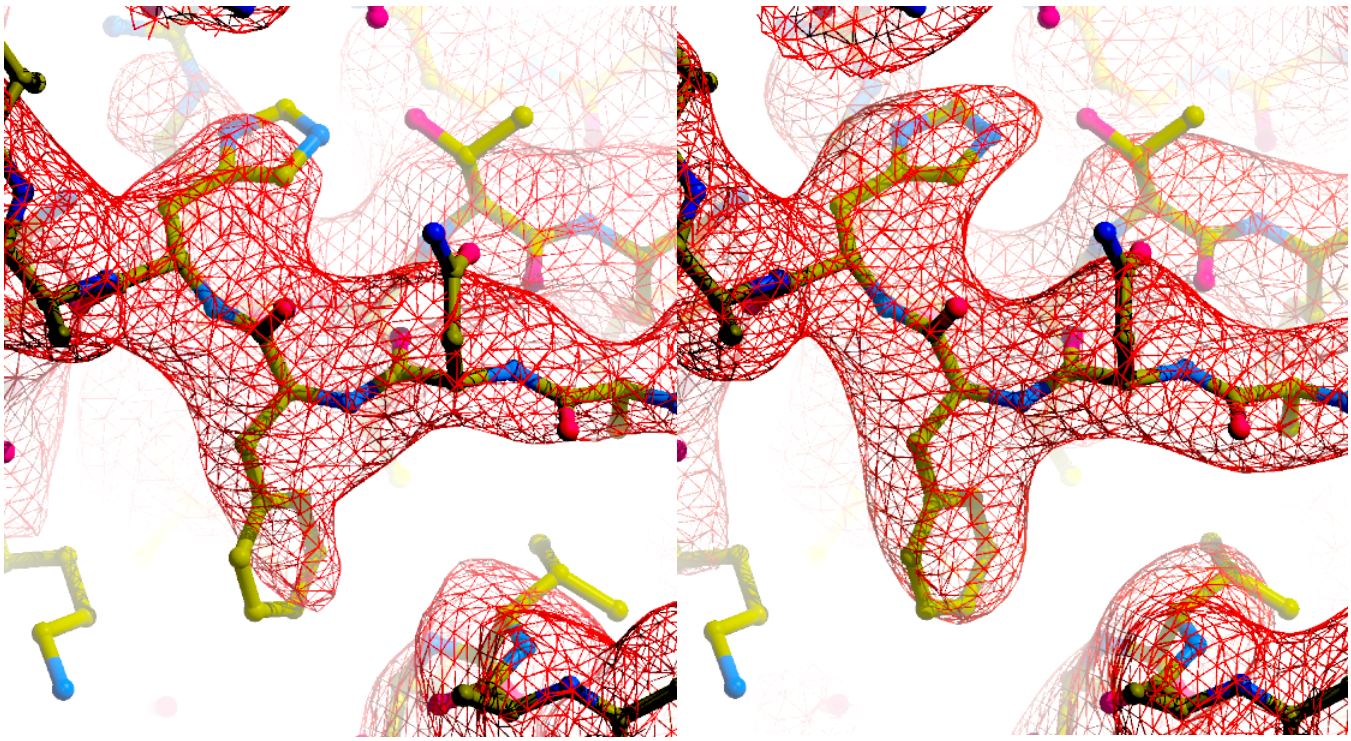


Figure 1: A small section through a map for EMDB entry 6344 along with the corresponding PDB entry (3jad) (left), and the results of automatic map sharpening (right).

to blurring. In our test cases, this procedure had little effect on the outcome so it is not the default.

Using the auto-sharpen procedure is easy. You just supply the map and the resolution. Figure 1 shows a small section through a map for EMDB entry 6344 along with the corresponding PDB entry (3jad) (left), and the results of automatic map sharpening (right). For local sharpening, you can optionally supply as well solvent content or a sequence file or the molecular mass of the molecule so that the regions of the map corresponding to the macromolecule can be identified. The solvent content will be guessed if not supplied.

You can see the methods in `phenix.auto_sharpen` in detail at (Terwilliger *et al.*, 2018).

Automatic segmentation of a map with phenix.segment_and_split_map

The next step in interpreting a cryo-EM map after sharpening is to segment the map. The purpose of segmentation is to break the map up into smaller pieces that are suitable for model-building. If the map has symmetry, segmentation can identify the basic unit of the map so that the same part of the map does not have to be interpreted many times. If the map has regions that are different chain types, segmentation can separate them and work with them individually.

There are three steps in segmenting a map with `phenix.segment_and_split_map`. The first is to contour the map and find all the regions defined by these contours. The next is to find the symmetry in the map. The third is to find a unique and compact set of regions that represents

the basic unit of the map. Applying symmetry to this basic unit of the map then can regenerate the entire map.

The `phenix.segment_and_split_map` tool contours the map using the same approach as was used above in auto-sharpening. The contour level is chosen to enclose about 20% of the volume representing the macromolecule but is adjusted further to enclose regions that typically correspond to about 50 residues in length. As symmetry-related parts of the map have the same densities, this procedure yields a set of regions that can be grouped in the next step based on the symmetry in the map.

The `phenix.segment_and_split_map` tool can identify symmetry in a map by comparing density at symmetry-related points. You can specify for example that the map has icosahedral symmetry, and the tool will test various common settings of this symmetry to see if they are present. Alternatively the tool can look through all common symmetries and settings to find the one with the highest symmetry that fits the map. You can also supply symmetry with a *Phenix* `ncs_spec` file or in BIOMTR records.

Using the symmetry in the map, the `phenix.segment_and_split_map` tool identifies the basic unit of the map. This is done by grouping the contoured regions in the map into symmetry-related sets and choosing a set of regions that forms a compact group. You can optionally surround this basic unit of the map with neighboring groups so that you have a better chance of enclosing a complete molecule in the basic unit of the map.

The tool writes out several map files. One is just the input map, shifted so that the (0,0,0) grid point of the map is at the coordinates (0. ,0. ,0.). Another is a map representing the basic unit of the original map, typically written to the file `box_map_au.ccp4`. This map is normally much smaller than the original map, particularly if there is a high degree of symmetry in the map. This map is shifted (again) so that its origin is again at (0,0,0). You can use this map for model-building if you want. You can even automatically shift the resulting model back to the position of the original map. Another set of maps are region maps, one corresponding to each unique region identified by `phenix.segment_and_split_map`. These maps all superimpose on the shifted input map, so their origins are not normally at (0,0,0).

Combining partial models with phenix.combine_models

The `phenix.combine_models` tool can combine the best parts of two or more models. You can use it in either of two different ways. One approach keeps segments in the models intact, and the second applies crossovers within segments.

In the first approach (`merge_by_segment_correlation=True`), each fragment in each model is scored based on correlation to the map. The scoring also includes a weight based on the square root of the length of the segment. It also includes weights based on whether a segment has been assigned to the sequence and the number of secondary structure hydrogen bonds in that segment. This approach can combine chains of different types as well. A final increment is added to the score for a segment if it is considered very likely to be correct with different thresholds for chains of different types.

Then the segments are picked in rank order to create a composite model. Any part of a segment that does not overlap an existing part of the composite model is kept. If symmetry is present, overlaps include that symmetry.

In the second approach, (`merge_by_segment_correlation=False`), a working model is created by taking all of the first model supplied and filling in any empty regions with fragments from other models. Then one by one, the segments in the working model are recombined with all other segments. To carry out recombination between two chains, residues that match in the two chains are identified. Then for each segment between matching pairs of residues, whichever chain has the higher correlation to the map is kept to create a composite model.

Removing poor parts of a protein-RNA model with phenix.remove_poor_fragments

When carrying out model-building for complexes between protein and RNA it sometimes happens that protein can be built accidentally into regions that are really RNA. The `phenix.remove_poor_fragments` tool is used to try and identify such incorrectly-built regions. The key idea in this approach is that these incorrectly-built regions tend to be isolated (not in the middle of a protein domain) and to have lower map-model correlation than the correct parts of the protein chains. The `phenix.remove_poor_fragments` tool ranks all protein segments on these criteria and removes the worst ones. The threshold that is chosen based on the number of residues of RNA expected in the molecule that were not built, the fraction of protein that was built, and the quality of the model, using the formula:

$$\texttt{residues\_to\_remove} = A * \texttt{protein\_built} * \texttt{rna\_not\_built}/(\texttt{cc}*\texttt{protein\_present}) \quad (1)$$

where `cc` is the map-model correlation for the protein part of the model, `protein_present` is the number of protein residues in the sequence file, `protein_built` are residues of protein built, and `rna_not_built` are residues of RNA in the sequence file minus the number built.

The logic of this formula is that there is a certain volume, proportional to `rna_not_built`, that is really RNA that might be accidentally built as protein. Furthermore the logic is that the number of residues of protein that might be built in that volume is higher if more residues of protein have been built and higher if the model-map correlation for the protein model is low. These relationships were seen in an analysis of ribosome structures. A scale factor A of 0.53 relating the optimal number of residues to remove to the other factors was found empirically.

Build a model with phenix.map_to_model

You can carry out fully automatic map interpretation and model-building with `phenix.map_to_model`. This tool first sharpens the map and carries out segmentation of the map to produce a map with the unique part of the original map and maps for each unique region in the map. The tool then carries out model-building both in the entire unique part of the map and in each individual region. Then `phenix.map_to_model` combines the best parts of all these models, applies symmetry, and refines the full model (see figure 2 for an example)

The `phenix.map_to_model` tool uses three different methods for model-building. One is standard resolve model-building, as in `phenix.autobuild`. This is applied to the entire unique

Figure 2: Example of a combined model.

part of the map. A second is finding regular secondary structure with `phenix.find_helices_strands`, also carried out on the entire unique portion of the map. The third is chain-tracing followed by refinement using secondary-structure restraints. This third method uses the trace_chain algorithm in `phenix.find_helices_strands`, which traces tubes of density by finding positions in the density that are separated by about 3.8 A. These C-alpha positions are converted to a main-chain model with Pulchra(Rotkiewicz & Skolnick, 2008). Then it uses the phenix tool `phenix.iterative_ss_refine`, which identifies secondary structure in a poor model, generates restraints based on an idealized version of that secondary structure, and refines with `phenix.real_space_refine` including those restraints. Each of these three approaches generates one preliminary model which is refined with `phenix.real_space_refine`. (These models are typically called `model_standard_PROTEIN.pdb`, `model_init_PROTEIN.pdb`, and model `_helices_strands_only_PROTEIN.pdb` respectively).

If there is RNA or DNA present in the structure (as guessed from the sequence file or specified by the user), RNA or DNA model-building is carried out in much the same way as was done for protein, except that there is no equivalent of the chain tracing algorithm used.

The preliminary models are then combined with the `phenix.combine_models` tool using the `merge_by_segment_correlation=True` approach. If there is symmetry present, the symmetry is included when combining models to ensure that there is no overlap. The resulting model is refined with `phenix.real_space_refine` to yield `model_PROTEIN.pdb`. If there is RNA or DNA present, those models will be in `model_RNA.pdb` or `model_DNA.pdb`.

To combine models with RNA, protein and DNA, the tool `phenix.combine_models` is used once again with the `merge_by_segment_correlation=True` approach. Each segment (of any chain type) is scored by length and map correlation and by presence of hydrogen bonds, and the best-scoring segment is picked for each part of the map.

In cases where RNA is a substantial part of the structure (at least 1/6 of the total), a final step is carried out in which poorly-fitting parts of the protein model are removed. This step is helpful for structures such as ribosomes where much of the structure is RNA, but where protein can sometimes be built accidentally into regions that are RNA but were not built as RNA. The tool `phenix.remove_poor_fragments` guesses how much of the protein part of a model might be built incorrectly and removes the appropriate amount of the worst-fitting and isolated parts of the protein part of the model.

Finally, any symmetry is applied to the model, parts that overlap by symmetry are removed, and the final model is refined with `phenix.real_space_refine`. The entire model is shifted to the original position of the original map and written out as `map_to_model.pdb` and the unique part of the model is also written out if symmetry is present (`map_to_model_unique.pdb`).

References:

Rotkiewicz, P. & Skolnick, J. (2008). *J. Comput. Chem.* **29**, 1460–1465.

Terwilliger, T. C., Sobolev, O., Afonine, P. V. & Adams, P. D. (2018). *BioRxiv*. 247049. https://www.biorxiv.org/content/early/2018/01/11/247049

Gemmi – new MX library

Marcin Wojdyr
Global Phasing Ltd, Cambridge, UK
Correspondence email: wojdyr@gmail.com

Introduction

At the beginning of 2017 CCP4 and Global Phasing Ltd. started a new software project – an open-source library called Gemmi[1] – funding one person (me) to work on it full-time for three years. The library is written in C++, with bindings for Python and Fortran. The project, agreed upon a few years ago, was largely motivated by the need to improve mmCIF handling in Refmac (Murshudov *et al*, 2011) and BUSTER (Bricogne *et al*, 2017) – refinement programs from the involved organizations. But the scope of the library is not limited to this. The scope, still flexible and driven by the needs of involved parties, can be split into handling of:

- macromolecular models (coordinate and restraint files),
- data on a 3D grid (CCP4 file format for maps and masks),
- and reflections (MTZ and SF mmCIF formats).

To implement the above points the library also needs to handle:

- the CIF syntax,
- and crystallographic symmetry.

All the listed areas include groundwork, such as reading and writing file formats, and higher-level functionality added on request from application developers. As an example of the latter, recently the 3D grid gained functions needed to determine the bulk solvent area.

The library is far from being finished and here I write only about a couple things that I have learnt during the first year of development.

[1] https://project-gemmi.github.io/

mmCIF and mmJSON

mmCIF format, the primary format used by the wwPDB, is not well suited for web applications. At least that is what RCSB, PDBe and PDBj think, as each of them developed own file format (MMTF, Binary CIF and mmJSON, respectively) for use in web-based molecular viewers. MMTF is a binary format with a complex compression scheme that reduces the data transferred from the server by a factor of four. Binary CIF is similar to MMTF, but preserves all the metadata from mmCIF at the expense of slightly larger size.

mmJSON is quite different and I would argue it is the most practical coordinate format for most of applications. It is a simple translation of the mmCIF content into JSON. Tables as stored as columns (`"tagA": ["value1", "value2"], "tagB": [1, 2]`). As a side-effect of writing data columns-wise, the transported (gzip-compressed) data is reduced by about 40%. More importantly, the JSON format can be easily read and written in any programming language. Of course CIF precedes JSON by a decade, and it was historically a good choice. But today we can also see the cost (in man-years across multiple projects) of a format with a niche syntax. So even now, 25 years after the CIF syntax was invented, adding mmJSON to the formats in the wwPDB archive (currently: PDB, mmCIF and PDBML) would make many future projects easier.

That being said, I wrote yet another parser of the CIF 1.1 syntax, basing on a library called `PEGTL`. To my best knowledge, so far the `cif_api` library had the fastest CIF parser (Bollinger, 2016), reportedly several times faster than `iotbx`. The Gemmi parser is 3x faster than `cif_api` (at least in the hands of the author). At the same time the fastest JSON parsers, hand-coded and carefully optimized, can handle hundreds of MB/s, several

times more than my CIF parser. So there is still plenty of room for optimization.

One way to utilize the speed of a CIF parser is to do a PDB-wide analysis on mmCIF files. For this I made a small utility (`gemmi-grep`) that prints values of selected tags in CIF files, reading input at the rate of about 100MB/s. It can go through a local copy of the wwPDB archive in half an hour, using a single processor core and uncompressing files on the fly. The tool was used to gather various statistics from the PDB, such as MX software statistics[2].

Space group settings

Mapping space group notation to a set of operations has been coded many times before, but I still pondered:

1) What data should be tabulated and what should be recalculated?
2) What space group settings should be included?

Question 1 is approached differently by different programs. Each space group settings must come with either the full list of operations, or with a minimal set of generators for the group, or with something between (for example centering vectors can be listed separately to save space). Then the operations can be encoded in various ways. After considering all the possibilities I went with the approach from SgInfo and sgtbx (and clipper in CCP4) – generating operations from the overly implicit Hall notation.

Regarding question 2, `SgInfo/sgtbx` (Grosse-Kunstleve *et al*, 2002) and International Tables for Crystallography (ITfC) vol. B[3] have a list of 530 settings (527 distinct settings; three settings in the group 68 are given with two different names). This list is used by many programs, notably by `Spglib`, which seems to be more popular than sgtbx among physicists.

CCP4 uses a file called `syminfo.lib` that is based on an older file called `symop.lib` and the data from `sgtbx`. It includes the 530 settings and a few more. Similarly, the OpenBabel library also has a file listing symmetry settings and operations, with 530 + 14 settings. The extra 14 came from the COD database. Some of them are described in "*A Hypertext Book of Crystallographic Space Group Diagrams and Tables*"[4] (for example C-centered tetragonal space groups) and mentioned in the latest edition of ITfC vol. A[5] (which, unlike the vol. B, uses new naming, e.g. "C -4 2 g1"). All the differences can be confusing, but have little practical importance in MX software. So for now Gemmi just includes the settings from `syminfo.lib`.

Adding new space group settings requires coming up with a Hall symbol. ITfC vol. B writes about an unambiguous method to select the Hall symbol. The method involves sorting operations into "a strictly prescribed order based on the shape of their Seitz matrices", but the details seem to be gone from the Internet. In this situation I manually picked Hall symbols for settings absent in ITfC vol. B.

Technicalities

Gemmi is written in C++ (no Python, although most of the functions will have Python bindings). For the sake of portability it uses C++11, ignoring new features of C++14 and 17.

Many C++ libraries are header-only, which makes them trivial to install. So far Gemmi is also a header-only library, but it may change as the library grows. Still, some parts (CIF parser, symmetry library) will stay header-only. This makes integration much easier. If a program only

[2] https://project-gemmi.github.io/pdb-stats/

[3](International Tables) Volume B home page. *urn:isbn:978-1-4020-8205-4* doi:10.1107/9780955360206000108

[4] http://img.chem.ucl.ac.uk/sgp/mainmenu.htm

[5](International Tables) Volume A home page. *urn:isbn:978-0-470-97423-0* doi:10.1107/9780955360206000114

needs to know space group operations, one can just copy a single file (gemmi/symmetry.hpp) to their project, and that is all.

Python bindings are generated with pybind11, which is inspired by Boost.Python but is smaller, faster to compile and easier to use. Python bindings work with Python 2.7 and 3.x, with CPython and PyPy. The latest development version can be downloaded, compiled and installed with a single pip command:

```
pip install git+https://github.com/project-gemmi/gemmi.git
```

We also need bindings for Fortran 2003; the work on it has just started.

Finally, as we all observe more and more software moving to web browsers, it would be nice to port Gemmi, or parts of it, to either JavaScript or WebAssembly. This would benefit some of the existing JavaScript programs, in particular UglyMol. The relatively small size of the C++ code base should help, but the details are yet to be investigated.

Acknowledgements

The project is funded by Global Phasing Ltd. and CCP4, overseen by Eugene Krissinel, Garib Murshudov and Gerard Bricogne, and was helpfully discussed with many other members of CCP4 and GPhL (in particular with Andrey Lebedev, Claus Flensburg, Clemens Vonrhein).

References

Murshudov GN, Skubák P, Lebedev AA, Pannu NS, Steiner RA, Nicholls RA, Winn MD, Long F, Vagin AA (2011). *Acta Cryst. D*: **67**, 355

Bricogne G, Blanc E, Brandl M, Flensburg C, Keller P, Paciorek W, Roversi P, Sharff A, Smart OS, Vonrhein C, Womack TO (2017). Cambridge, United Kingdom: Global Phasing Ltd.

Bollinger JC (2016), *J. Appl. Cryst.* (2016). **49**, 28

Grosse-Kunstleve RW, Sauter NK, Moriarty NW, Adams PD (2010). *J. Appl. Cryst.* **35**, 126

Overfitting to Ramachandran and geometry criteria in the cryoEM Model Challenge

Christopher J Williams, David C. Richardson, and Jane S Richardson
Duke University

We are interested in the extent and effects of overfitting to validation criteria, and the EM Model Challenge[1] provided a useful dataset to analyze overfitting. The dataset was suitable because, for each modeler group, it contains multiple depositions of very different structures using very similar methods. This allows us to construct a better sense of each group's overall behavior. However, many modelers are willing – quite reasonably – to skip over loops and other difficult regions. This prevents us from constructing a truly complete picture of how their modeling tools behave.

Overfitting to Ramchandran

To assess overfitting to Ramachandran criteria, for each modeler group, we accumulated all the Ramachandran points from their first model for each target into a single distribution. The resulting Ramachandran distributions are strikingly varied. Appropriately to a modeling challenge, the modelers seem to have been very aggressive in using diverse methods. However, almost all of these methods have resulted in some degree of overfitting to Ramachandran criteria.

Figure 1 shows typical patterns of Ramachandran overfitting within the cryoEM Challenge dataset. Severals groups restrained their Ramachandran distribution to a different – and much stricter – set of contours than our Top8000-derived set (Richardson 2013). Group EM119 (1a) shows this alternate distribution most strongly. Group EM120 (1b) also shows restraint to contours

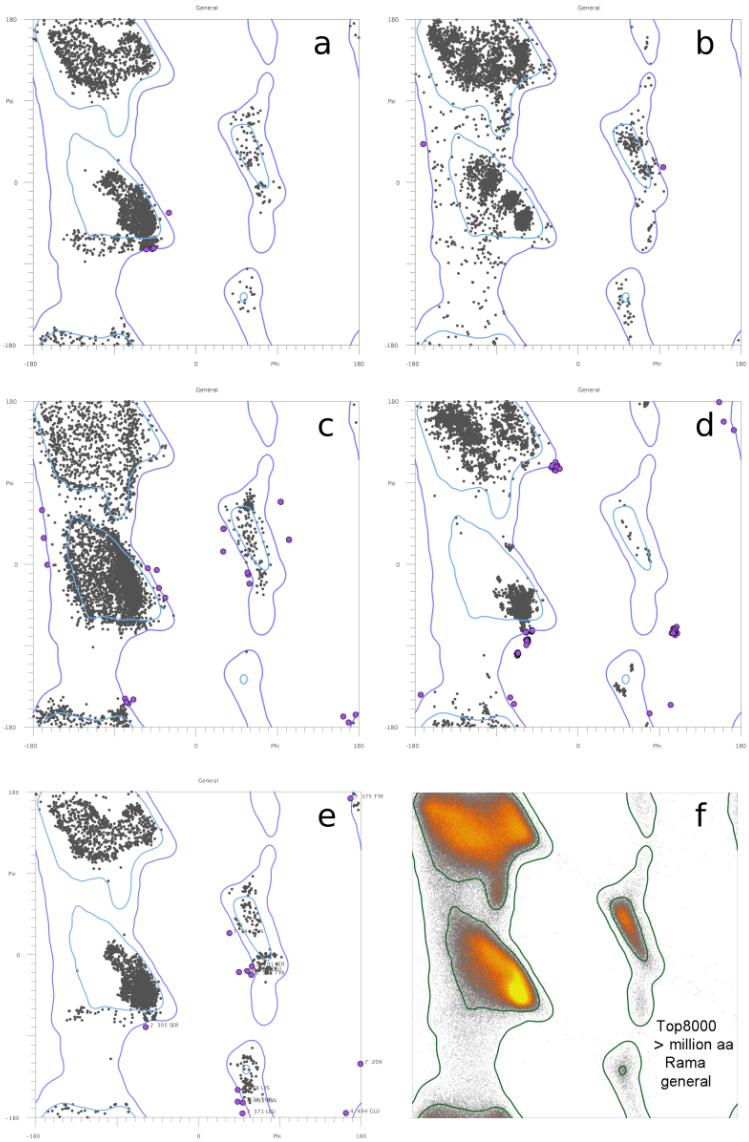


Figure 1: Ramachandran diagrams showing unusual distributions. Ramachandran distributions of all general-case residues from cryoEM Challenge groups EM119 (a), EM120 (b), EM130 (c), and EM189 (d) show various types and degrees of overfitting to Ramachandran criteria. The Ramachandran distribution from a deposited PDB structure, 3ja8, shows a similar pattern of overfitting (e). Our standard Top8000 Ramachandran distribution, with heat-mapped population density, is provided for comparison (f).

[1](http://challenges.emdatabank.org/?q=model_challenge)

within the *Favored* bounds, with the addition of alternating patches of highly-populated and empty regions. Group EM130 (1c) shows an edge-tracing effect, where the edge of the *Favored* region is unusually highly populated, possibly due to Ramachandran restraints moving *Allowed* residues until they cross just into the *Favored* region, or possibly due to an energy that particularly favors certain atomic contact distances. Group EM189 (1d) shows an ultra-restrained alpha helix region, which might be an extreme version of the patchy regions in 1b. Our standard Ramachandran distribution, with heat-mapped population density, is provided for comparison (1f).

We had expected the edge-tracing phenomenon in 1c to be relatively common, but in fact only a few groups showed significant tracing of our *Favored* contours. It should be noted, however, that EM119 (1a) does show an edge tracing effect at the bounds of their chosen contours. We also observed that many distributions (e.g. 1b and 1c) showed a fairly hard ϕ cutoff at about -60 degrees. This leaves the rightmost areas of both the alpha and the beta regions unnaturally empty.

These peculiar distributions are by no means limited to the sometimes-speculative software of the cryoEM Challenge, however. Figure 1e shows the Ramachandran distribution of 3ja8, a PDB-deposited EM structure from 2015. This distribution shares many characteristics with 1a, including tightened contours within the *Favored* region, an additional horizontal band below alpha, and a strong edge-tracing effect in beta. The overfitting that occurred during the cryoEM Challenge is also seen in final deposited cryoEM structures.

Overfitting of covalent geometry

We also assessed overfitting to covalent geometry criteria. For each group, we performed covalent geometry validation with mmtbx.mp_geo. For each mainchain bond or angle (those involving N, CA, C, O, and CB), we determined the percentage of the total population in each 1-standard-deviation bin from the ideal value, making no distinction between positive and negative deviations from ideal. If the covalent geometry deviations followed a normal distribution, the populations of the first three bins would be roughly 68, 27 and 4%. A deviation of 4σ or more is considered an outlier.

Figure 2 shows some representative distributions for covalent bond lengths on the left and angles on the right. Bond lengths were typically very highly restrained to within 1 sigma of ideal, as in EM119 and EM130, though some groups like EM120 clearly followed some other pattern. Bond angles were generally less tightly restrained and slightly closer to a normal distribution, see again EM119 and EM130, but real approximations of a normal distribution were very rare.

Effects of overfitting

Validation tools that modelers were not optimizing against, such as CaBLAM (Williams, 2018), give us suggestive evidence of structural harm caused by overfitting. Figure 3 shows one such example. Target T0002 (PDB 3j9i, a proteasome) contains a cluster of covalent geometry outliers (3a) around Ala107 of chain 1. In EM130's model, these geometry outliers have been resolved, but at the cost of introducing a Cα geometry outlier identified by CaBLAM (3b). The Cα geometry outlier is not very far from an allowed conformation, so permitting slightly more flexibility in the surrounding covalent geometry might prevent this distortion.

A more dramatic example of structural harm from overfitting is the placement or refinement of Ramachandran points into the wrong favored regions, which we have observed in both the cryoEM Challenge models and in the PDB. Figure 4 shows an alpha helix from PDB 3ja8 with a clear and severe modeling error at Leu234 (a). This modeling error placed the Ramachandran point for Leu234 in or near the beta region (b), and refinement moved it into the cluster at the edge of

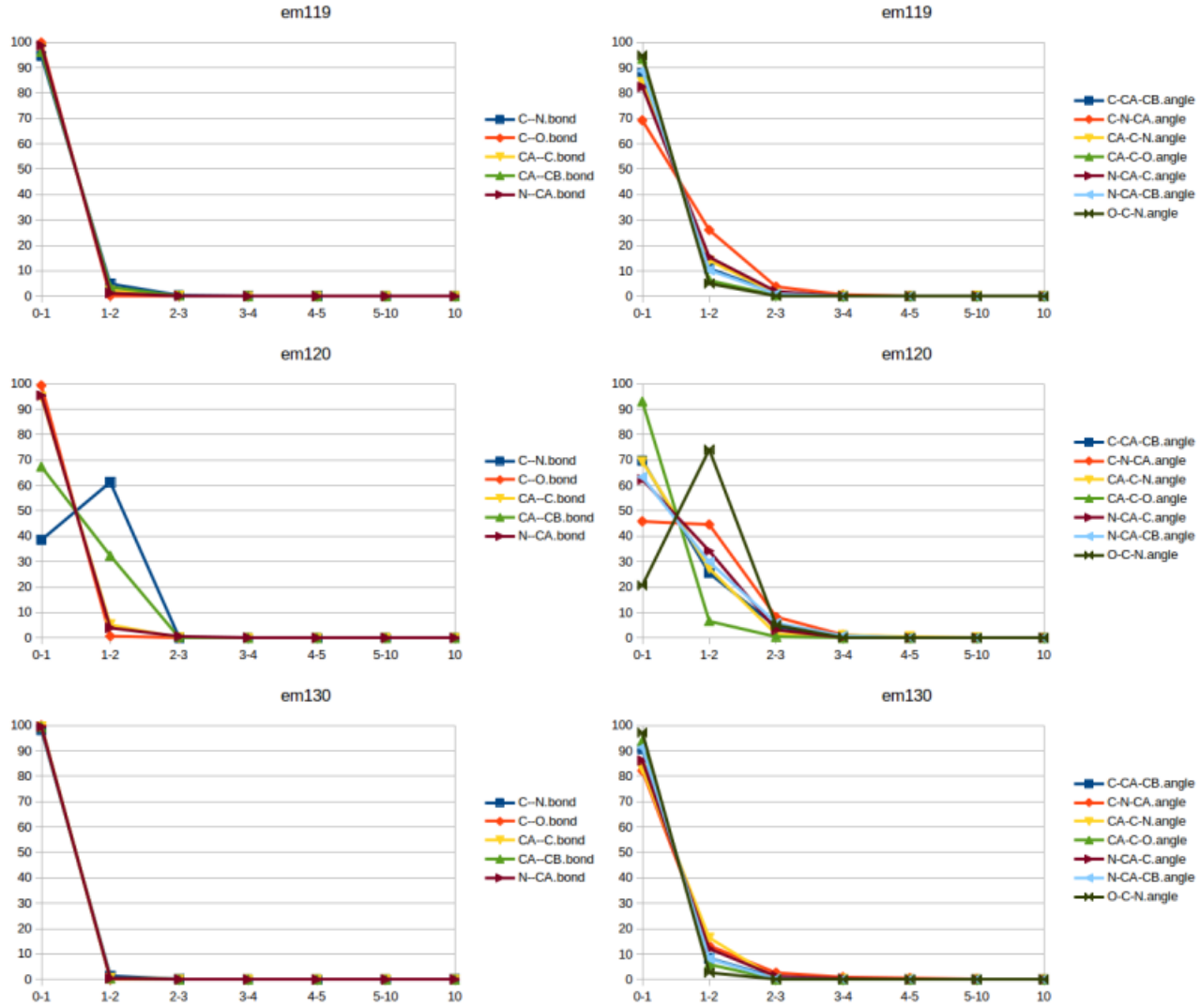


Figure 2: Population distributions for covalent bond geometry of cryoEM Challenge models. X-axis is number of standard deviations from ideal (4 is the outlier cutoff), y-axis is percent of population. Groups EM119 and EM130 show highly restrained distributions typical of the challenge models. Group EM120 shows an unusual distribution suggestive of either modeling errors or a mismatch between geometry assumptions.

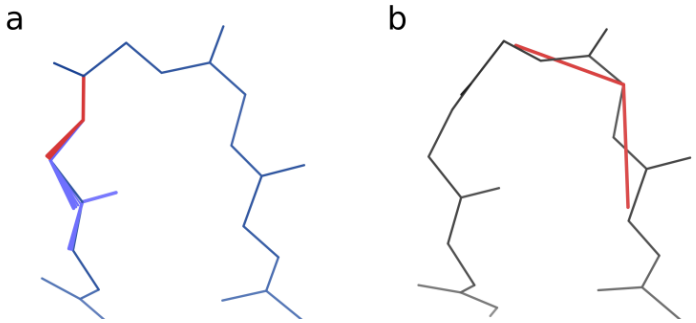


Figure 3: Resolution and introduction of geometry outliers in cryoEM Challenge target T0002 by group EM130. The target (a) contains severe bond angle outliers around chain 1 Ala107. The model (b) resolves these outliers but introduces a Cα geometry outlier – probably an unrealistic Cα virtual angle – in the process. The arrangement of the outliers suggests but cannot prove that over-idealization of covalent geometry lead to distorted Cα geometry in the model.

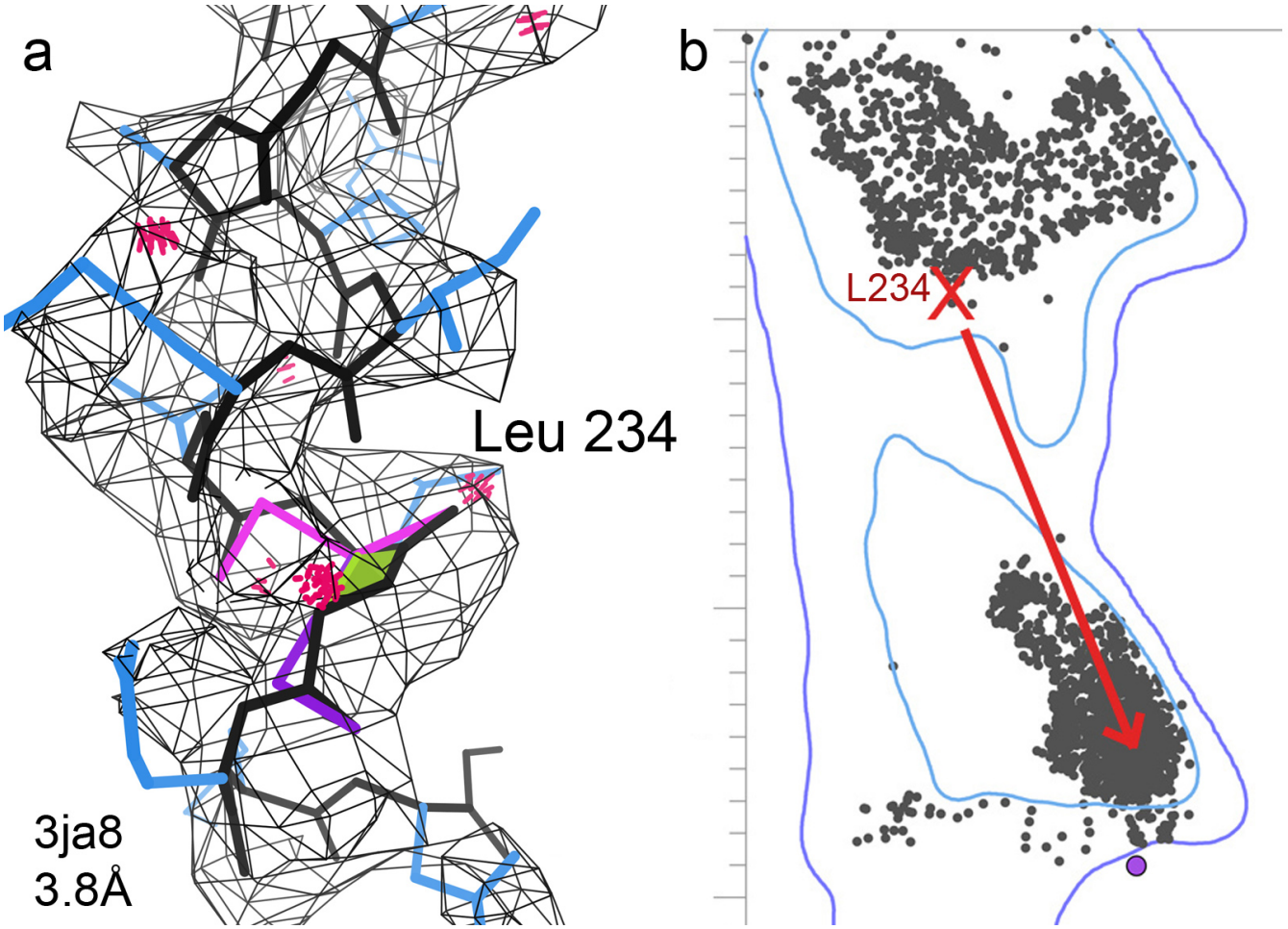


Figure 4: Compounding and obfuscation of a secondary structure error in 3ja8.pdb. An incorrect cis-peptide and distorted alpha helix (a) placed the Ramachandran point for Leu234 near the beta region (b). During refinement, overfitting to Ramachandran criteria moved this point solidly into beta, rather than correcting the underlying modeling error.

the beta contours. Overfitting to Ramachandran criteria has thus compounded an existing error and hidden it from casual validation.

Discussion

Given the overfitting to Ramachandran criteria in the challenge models, the simultaneous heavy restraining of covalent geometry criteria came as a surprise. It should not be possible to distort the Ramachandran distribution in the ways seen in Figure 1 without consequences, and bond geometry seemed a likely place to dump the consequences. And indeed, the geometry distributions of group EM120 show the sort of off-ideal peaks we might expect if small geometry deviations were taking up the strain of an artificially tight Ramachandran distribution. However, EM120 is the exception rather than the rule, and bond geometries, especially bond lengths, are highly restrained for almost all of the challenge models. Both Ramachandran and geometric criteria are being overfit, and this overfitting affects the implicit or explicit balance of fit-to-map and fit-to-geometry in the refinement target function. As we work as a community to assess and address overfitting, one of the questions we will have to explore is: where else do the consequences go?

References

Richardson, JS, Keedy, DA, Richardson, DC (2013). "The Plot" Thickens: More Data, More Dimensions, More Uses. In *Biomolecular Forms and Functions: A Celebration of 50 Years of the Ramachandran Map* (pp. 46-61).

Williams, CJ, Headd, JJ, Moriarty, NW, Prisant, MG, Videau, LL, Deis, LN, Verma, V, Keedy, DA, Hintze, BJ, Chen, VB, Jain, S, Lewis, SM, Arendall, WB, Snoeyink, J, Adams, PD, Lovell, SC, Richardson JS, Richardsdon DC. (2018) "MolProbity: More and better reference data for improved all-atom structure validation", *Protein Science*, **27**(1), 293-315.

Cβ deviations and other aspects in Amber vs CDL refinements

Jane Richardson and David Richardson, Duke University

Background

The Amber force-field (Case 2016) has been implemented for use in Phenix refinement. Nigel Moriarty and David Case have been tuning and extensively testing this new feature. In collaboration with that effort, the Duke Phenix team is looking at individual results from paired refinements that use Amber vs. the current default refinement that includes the CDL (Conformation-Dependent Library; Moriarty 2014). This article describes our most interesting results so far.

Analysis of parallel refinement results for 1xgo

The 1xgo example at 3.5Å resolution is an ideal test case for our purposes because there is an essentially identical structure available at high resolution to define the correct answers for local conformations. 1xgo ended up scoring somewhat better on most measures after Amber refinement than after CDL refinement, as seen in the MolProbity-chart stoplight comparisons in Figure 1, which also shows the original deposited 1xgo (Tahirov 1998) and the 1.75Å 1xgs structure done at the same time of the same molecule in a different space group (Tahirov 1998). The CDL refinement completely idealizes all the covalent geometry with tight restraints, rotamers are about equal and Amber does better on both Ramachandran and clashscore.

However, there was concern that six residues had Cβ-deviation outliers (≥0.25Å) after Amber refinement and whether those reflected any issues with the force-field terms. Figure 2 below shows the "bullseye" Cβ deviation plots for all four coordinate sets, centered at the ideal Cβ and with outliers emphasized. The CDL-refine plot is tight to an extreme degree, perhaps justifiable at low resolution but debatable. Even without an artificial Cβdev restraint, if Tau (N-Cα-C), N-Cα-Cβ and C-Cα-Cβ angles are all tightly restrained there will be no Cβdev outliers. The Amber plot is a bit on the loose side, but shows a good scatter with no evidence of target anomalies and extends farthest out in the direction that is indeed most vulnerable. The crucial issue is whether those outliers are just being allowed by too-lenient restraints or are diagnosing real

MOL PROBITY

Duke Biochemistry
Duke University School of Medicine

Summary statistics

		1xgo 3.5Å			CDL-refine			Amber-refine			1xgs 1.75Å		
All-Atom Contacts	Clashscore, all atoms:	78.83		16th	7.42		100th	0.42		100th	6.78		90th percentile
	Clashscore is the number of serious steric overlaps (> 0.4 Å) per 1000 atoms.												
Protein Geometry	Poor rotamers	65	26.75%		46	18.93%		50	20.58%		15	3.09%	
	Favored rotamers	133	54.73%		158	65.02%		156	64.20%		445	91.56%	
	Ramachandran outliers	24	8.19%		12	4.10%		11	3.75%		0	0.00%	
	Ramachandran favored	204	69.62%		239	81.57%		256	87.37%		573	97.78%	
	MolProbity score^	4.29		13th	3.09		85th	2.26		99th	1.79		77th percentile
	Cβ deviations >0.25Å	2	0.73%		0	0.00%		6	2.20%		2	0.37%	
	Bad bonds:	0 / 2353	0.00%		0 / 2353	0.00%		0 / 2353	0.00%		0 / 4706	0.00%	
	Bad angles:	11 / 3178	0.35%		0 / 3178	0.00%		29 / 3178	0.91%		3 / 6356	0.05%	
Peptide Omegas	Cis Prolines:	1 / 14	7.14%		1 / 14	7.14%		1 / 14	7.14%		2 / 28	7.14%	
	Twisted Peptides:							1 / 294	0.34%				
Low-resolution Criteria	CaBLAM outliers	15	5.14%		17	5.82%		11	3.77%		5	0.86%	
	CA Geometry outliers	6	2.05%		5	1.71%		7	2.40%		2	0.34%	

Figure 1: Comparison of MolProbity summary validation for 1xgo, CDL refinement, Amber refinement and 1xgs.

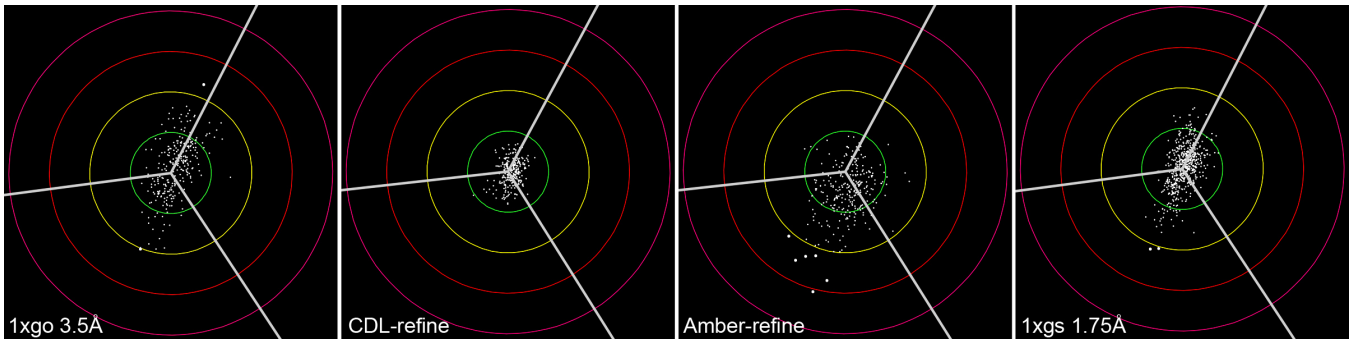


Figure 2: Comparison of "bullseye" plots of Cβ deviation for 1xgo, CDL refinement, Amber refinement and 1xgs.

problems – so we looked at them all and compared each with our expectations, with electron density and especially with the 1xgs 1.75Å structure.

Of course we will need to analyze more cases, but it is remarkable for 6 arbitrary examples of some effect to all point consistently and convincingly to the same conclusion. Each one of the outlier residues has a real problem uncorrectable by downhill refinement, with either the sidechain or the backbone or both in the wrong local-minimum conformation. This flagging of outliers is the most important purpose of validation: to support either people or software in testing distinct alternatives to correct conformational misfittings that need not just a nudge but a big change.

The 4-part comparison in Figure 3 illustrates Leu 204, the simplest of the 6 Amber Cβdevs, where the rotamer is wrong and pure refinement cannot correct it. The original model-building for 1xgo tried too hard to push all atoms into the density, resulting in a curled-up sidechain with a near-zero chi1 angle. The high-resolution 1xgs structure has completely unambiguous density showing Leu 204 with an **mt** conformation, the overwhelmingly most common Leu rotamer (Hintze 2016). Both refinements shifted enough to lose the clash that was in 1xgo, but the Cβdev from Amber, as well as the rotamer outlier, signal even at 3.5A that this sidechain needs to be rebuilt.

Alanine is seldom a Cβdev outlier, but the well-ordered 3.5Å electron density for the Ile-Gly-Ala194-Gly semi-extended sequence just does not have enough information content to specify its conformation correctly. As seen in Figure 4, both refinements resolve the clashes in 1xgo, but do not improve the backbone conformation, as reported by CaBLAM outliers (Williams 2018) and by Amber's Ala Cβdev. With all backbone COs clear in the density at 1.75Å, 1xgs identifies a favorable, unambiguously correct conformation. To achieve that, ϕ,ψ angles change by at least 30°

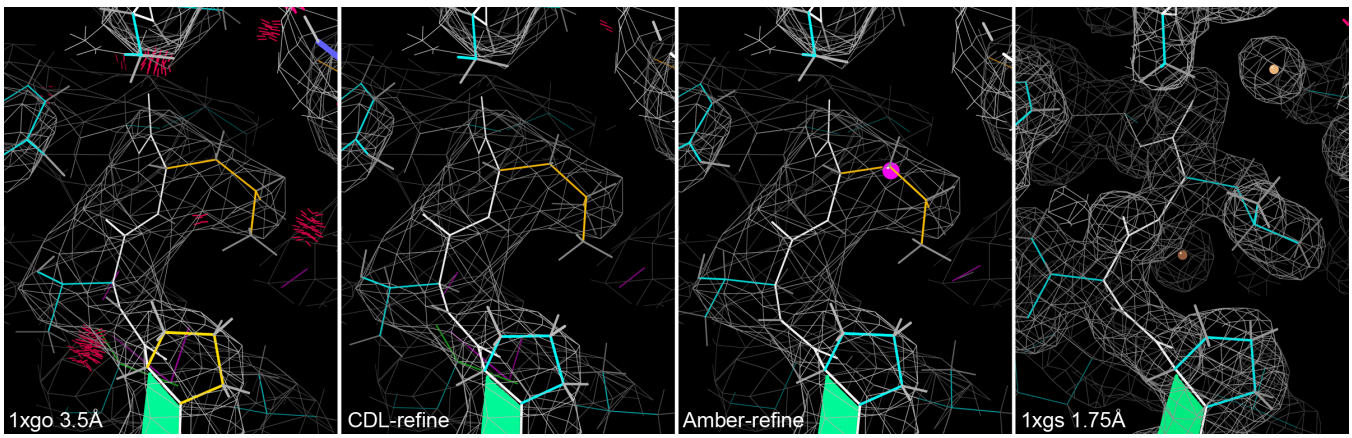


Figure 3: Comparison of Leu 204 rotamer outlier for 1xgo, CDL refinement, Amber refinement and 1xgs.

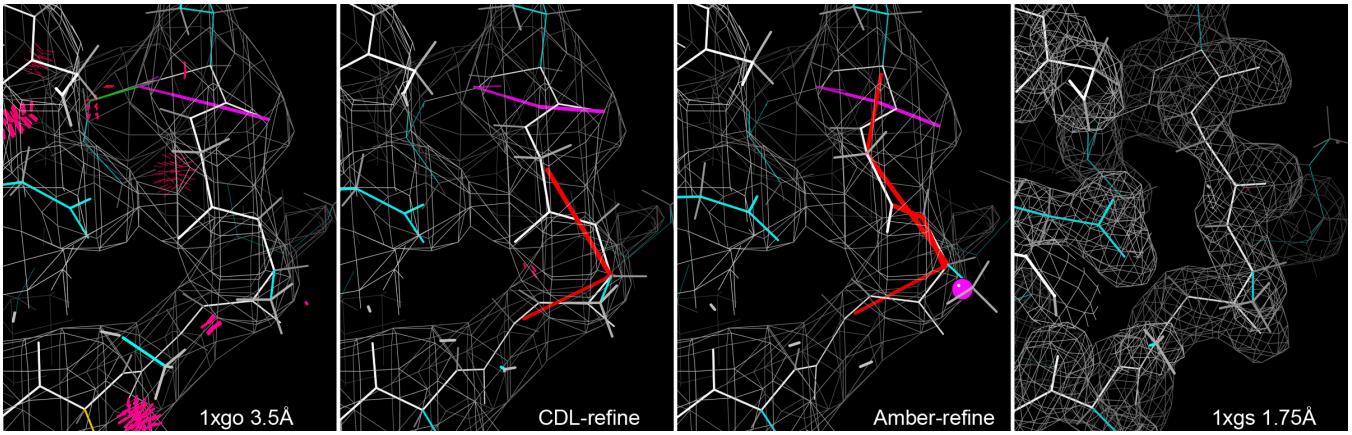


Figure 4: Comparison of Ile-Gly-Ala194-Gly backbone for 1xgo, CDL refinement, Amber refinement and 1xgs.

in each of the 4 residuesand ψ of Gly193 and ϕ of Ala194 change more than 120°.

For Leu 253 on a helix, both sidechain rotamer and backbone shifts are involved (see Figure 5). In the original 1xgo model, Leu 253 is in the very low-population **pt** rotamer, which unsurprisingly is maintained in both refinements although slightly shifted. However, the 1xgs structure shows very clear density for the highest-population **mt** rotamer and that conformation looks as though it would have fit the low-resolution density just as (moderately) well as **pt** did. The helix conformation is very non-ideal in 1xgo, with a distorted overall shape and a Ramachandran outlier. Both refinements resolve the numerous clashes while rotamer outliers stay similar. CDL refinement does not move the backbone; Amber improves it toward ideality, although still not as regular as shown by the high-resolution 1xgs.

In order to understand better what each of these refinement protocols can and cannot do, our group will need to examine details where rotamers or clashes have improved with pure refinement. The questions are whether distinct conformations can ever be interconverted and in which circumstances the changes from outlier to allowed rotamer (or Ramachandran) have only shifted across a close borderline versus when they have indeed discovered the correct rotamer.

The bottom line

Our overall conclusion is that refinement cannot be expected to correct local conformations in the wrong local minimum, especially at low resolution. Ideally, as many as possible of those problems should be corrected early-on, either by

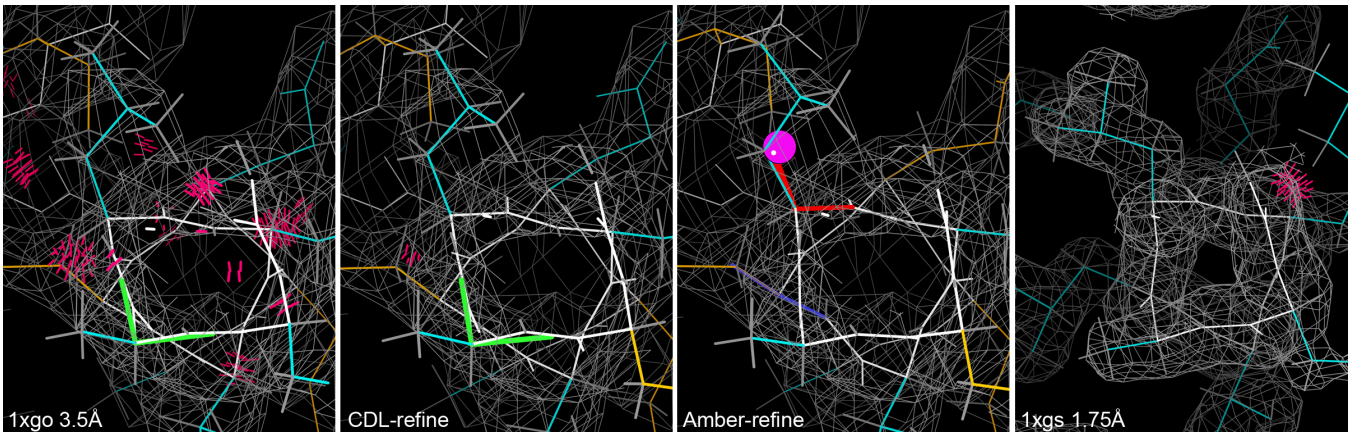


Figure 5: Comparison of Leu 253 on helix for 1xgo, CDL refinement, Amber refinement and 1xgs.

manual rebuilding or by an automated procedure that evaluates distinct alternatives. It is very counterproductive to sweep such problems under the rug by artificially adding terms that disallow validation outliers. The CDL refinement corrects all geometry outliers, preventing them from showing problems, but does leave backbone and some steric flags. The Amber refinement not only corrects clashes but improves van der Waals contacts (even among incorrect rotamers or backbone!), but allows geometry to report problems. A strategy partway in between might be even better. However, we feel very positive about the Amber target after looking at these details: if it were presented with a model where most big misfittings had been corrected, it would do an excellent job of tuning the details correctly. Refinement should work honestly on all independent single terms, while the whole point of combined, non-refined validation criteria (Rfree, rotamers, Ramachandran, Cbdev, CaBLAM, RNA backbone conformers and ribose puckers, …) is to flag places that probably need large changes between barrier-separated conformations. We should not try to hide those flags artificially, but should develop better procedures for using what they tell us.

References:

Case DA, Betz RM, Cerutti DS, Cheatham TE III, Darden TA, Duke RE, Giese TJ, Gohlke H, Goetz AW, Homeyer N, Izadi S, Janowski P, Kaus J, Kovalenko A, Lee TS, LeGrand S, Li P, Lin C, Luchko T, Luo R, Madej B, Mermelstein D, Merz KM, Monard G, Nguyen H, Nguyen HT, Omelyan I, Onufriev A, Roe DR, Roitberg A, Sagui C, Simmerling CL, Botello-Smith WM, Swails J, Walker RC, Wang J, Wolf RM, Wu X, Xiao L, Kollman PA (2016), AMBER 2016, University of California, San Francisco

Hintze BJ, Lewis SM, Richardson JS, Richardson DC (2016) "MolProbity's ultimate rotamer-library distributions for model validation", *Proteins: Struc Func Bioinf* **84**: 1177-1189

Moriarty NW, Adams PD, Karplus PA (2014) Details of the conformation-dependent library, *Comput Crystallogr Newsletter* **5**: 42-49

Tahirov TH, Oki H, Tsukihara T, Ogasahara K, Yutani K, Ogata K, Izu Y, Tsunasawa S, Kato I (1998) Methionine aminopeptidase from hyperthermophile *Pyrococcus furiosus, J Mol Biol* **284**: 101-124

Williams CJ, Hintze BJ, Headd JJ, Moriarty NW, Chen VB, Jain S, Prisant MG Lewis SM, Videau LL, Keedy DA, Deis LN, Arendall WB III, Verma V, Snoeyink JS, Adams PD, Lovell SC, Richardson JS, Richardson DC (2018) MolProbity: More and better reference data for improved all-atom structure validation, *Protein Science* **27**: 293–315

A few benchmark tests of various compilers on Linux and Windows.

Robert D Oeffner

Cambridge Institute for Medical Research, University of Cambridge, Cambridge Biomedical Campus
Wellcome Trust/MRC Building, Hills Road, Cambridge CB2 0XY

Correspondence email: rdo20@cam.ac.uk

Introduction

The Phenix crystallographic software suite is deployed on three different platforms; MacOS, Windows and Linux. In this article we present the results of a few benchmark tests of the Phaser executable produced by compilers from GNU, Intel and Microsoft. The platforms we test on are Ubuntu 16 and Windows 10 as well as Windows 10 with Windows Subsystem for Linux (WSL) running Ubuntu 16.

Motivation

We were interested to determine which platform and compiler provides the fastest performance of the Phenix software. Moreover, benchmark tests on the WSL platform are of interest given it allows running Linux executables directly on Windows without the overhead of starting up a virtual machine, but it might incur a performance cost.

We are interested in the overall time taken to run executables, i.e. the wall time. Phenix is a software suite consisting of python scripts and C++ code compiled into python modules. It is the choice of C++ compilers we test here. We focus on Phaser which, like the rest of Phenix, depends on the CCTBX library. During the time of performing the benchmark tests BIOS and OS patches for the "Meltdown" bug for Intel CPUs became available (see Patches in the appendix). Additional benchmark tests for the patched platforms have therefore been included.

Compilers and platforms

We tested the platforms and compilers listed in Table 1.

The specific versions of the operating systems are:

- Ubuntu 16.04.3 kernel 4.10.0-28-generic
- Windows 10, version 1709, build 16299.125
- Ubuntu 16.04.3 kernel 4.4.0-109-generic
- Windows 10, version 1709, build 16299.192

The first two operating systems in the above list were not patched for the Intel Meltdown bug whereas the last two have been patched for the Intel Meltdown bug as has the BIOS on the PC.

The Visual Studio 2008 compiler is used for Phenix Windows builds due to it using the same C-runtime as the python executable that is deployed with the python interpreter distributed from python.org.

The Visual Studio 2015 compiler is used for building Python 3 on Windows. As Phaser distributed with Phenix eventually will be migrated to Python 3 this compiler or newer ones are of interest.

The MinGW-W64 Gnu 5.3.0 compiler is used for building Phaser distributed with CCP4 version 7 on Windows. This is a Gnu compiler ported to Windows, which is of interest as it comes with the Gnu implementation of the C++ Standard Template Library and OpenMP library.

The Intel Parallel Studio XE 2018 for Linux compiler, unlike the others, is not free, but is reputed to produce very fast executables.

Table 1: List of compilers and platforms used for benchmarking. Abbreviations in bold are used in subsequent figures and tables denoting the executables produced by the respective compilers.

C++ Compiler	Platforms (64 bit)	Comment
Visual Studio 2008 (**VS2008**) by Microsoft	Windows 10	used for current Phenix build with python 2 on Windows
Visual Studio 2015 (**VS2015**) by Microsoft	Windows 10	to be used in future for Phenix build with python 3 on Windows
MinGW-W64 g++ version 5.3.0 (**MinGW 5.3.0**)	Windows 10	used for CCP4 Windows build of Phaser
Intel Parallel Studio XE 2018 for Linux (**Intel**)	Native Ubuntu 16.04, Ubuntu 16.04 on WSL on Windows 10	uses the C++ Standard Template Library from the Gnu compiler
g++ version 5.4.0 for Linux (**Gnu 5.4.0**)	Native Ubuntu 16.04 Ubuntu 16.04 on WSL on Windows 10	default on Ubuntu 16.04
g++ version 4.4.7 for Linux (**Gnu 4.4.7**)	Native Ubuntu 16.04 Ubuntu 16.04 on WSL on Windows 10	default on Centos 6, build of Phaser in Phenix

The Gnu 5.4.0 compiler suite for Linux is the default on Ubuntu 16.04. It can therefore be anticipated that a user of Phenix will use this compiler if rebuilding Phenix from sources on Ubuntu. Given the popularity of Ubuntu amongst other flavours of Linux this compiler version is therefore relevant to benchmark.

The g++ 4.4.7 compiler for Linux is what is currently used for building Phenix for Linux on the Centos6 platform. The combination of this compiler and platform is convenient as it permits Phenix to be installed on a variety of Linux platforms.

In total there are six different executables three of which will run both on WSL as well as native Linux. In the subsequent text executables produced by the Visual Studio or the MinGW compilers will be referred to as Windows executables or win32 executables. The Intel, the Gnu 5.4.0 and the Gnu 4.4.7 executables tested on native Linux will be tagged with "native". If they are tested on WSL they will be tagged with "WSL".

Hardware

The PC used for the test is an 8 core Intel Xeon CPU E5-1660 v3 @ 3.0GHz. Hyperthreading is enabled meaning that the operating systems see it as having 16 logical cores. The available memory is 32 Gb with L1, L2 and L3 cache memory of 512Kb, 2Mb and 20Mb respectively.

To benchmark on Windows 10 or on WSL the PC was booted with the installed Windows 10. To benchmark on native Ubuntu 16 it was booted with an Ubuntu 16.04.3 ISO image from an external USB device. This ensured the hardware was the same during all tests.

Patching the PC for the meltdown bug entails doing a BIOS update, a Windows update as well as updating the kernel of an existing

Ubuntu 16.04.3 installation on a different PC. The Linux LiveKit software available from https://www.linux-live.org/ was then used to deploy this patched Ubuntu version as a bootable image to an external USB device. This USB device could then be used for booting the PC into the patched version of Ubuntu 16.04.3.

Software

The software used for the tests is written in C++. The times were recorded with the Linux "time" command. The CPU usage is recorded with the Linux "top" command or on Windows with a custom written C# script using Windows Management Information templates obtained from https://www.microsoft.com/en-us/download/details.aspx?id=8572. We are interested in how execution speeds scale with the number of threads. The threading technology tested here is OpenMP. Given the hardware exposes 16 logical cores the maximum number of OpenMP threads tested for is 16.

When a program run in serial mode, i.e. only uses one CPU, the usage is 100%. But if it is allowed 16 OpenMP threads a CPU usage of up to 1600% may occur whenever an OpenMP parallelized loop is executed. In all the figures depicting execution times against number of OpenMP threads the axes have been set to logarithmic scale as a way of verifying the scalability of parallelization.

Benchmark Tests

The following sections demonstrate the differences in execution time achieved with executables produced with the different compilers and on different platforms as listed above. Each test program has been executed on the above platforms and execution times are presented as graphs and tables. The wobbles in some of the graphs are due to the PC occasional executing other uninterruptable tasks such as virus scanning.

The π-program

We tested a small program that calculates the natural number **π** through looping over a slowly converging Fourier series expansion that was made even slower by redundant use of exp(), cos() and sqrt() operations. It can take advantage of OpenMP parallelization as illustrated below in figure 11 in the appendix.

Benchmarks of the π-program on un-patched platforms

As is seen in figure 1 the program scales well with the number of OpenMP threads being used; data points for each compiler follow almost straight lines sloping down albeit not quite achieving half the execution time as the number of threads doubles.

It is apparent that the execution times can be categorised into three groups:

- Intel executable
- Gnu5.4.0, MinGW5.3.0 and VS executables,
- Gnu4.4.7 executable

As seen in table 2 the Intel executable clearly outperforms all other executables. This is whether or not the executable is run on native Linux or on WSL. The speed of executables built with MinGW5.3.0, VS2015, VS2008, Gnu5.4.0 WSL or Gnu5.4.0 native Linux are roughly the same, about twice as slow as the Intel executable. The speed of an executable built with Gnu4.4.7 running either on native Linux or on WSL is between three or five times slower than the Intel executable.

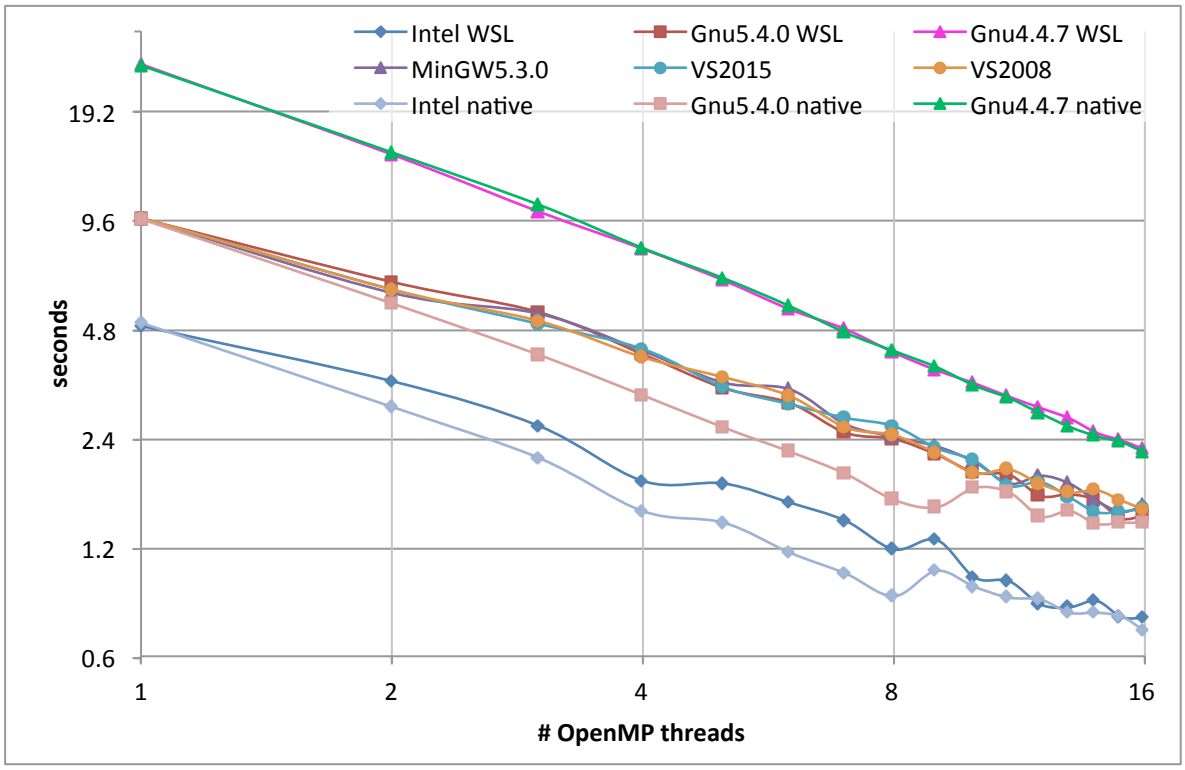


Figure 1: Graphs of execution times in seconds of the π calculation test job as a function of OpenMP threads for Phaser built with different compilers and running on different un-patched platforms.

Table 2: Execution times in seconds of the π calculation test job as a function of OpenMP threads for Phaser built with different compilers and running on different un-patched platforms.

OpenMP threads	Intel WSL	Gnu5.4.0 WSL	Gnu4.4.7 WSL	MinGW 5.3.0	VS2015	VS2008	Intel native	Gnu5.4.0 native	Gnu4.4.7 native
1	4.905	9.718	25.956	9.713	9.727	9.753	5.001	9.700	25.787
2	3.457	6.501	14.627	6.065	6.196	6.195	2.937	5.673	14.821
3	2.603	5.366	10.183	5.316	4.981	5.073	2.124	4.094	10.637
4	1.835	4.108	8.036	4.193	4.247	4.030	1.516	3.166	8.077
5	1.807	3.314	6.582	3.430	3.334	3.551	1.409	2.582	6.672
6	1.600	3.017	5.479	3.290	2.994	3.154	1.165	2.216	5.593
7	1.426	2.498	4.833	2.635	2.745	2.583	1.021	1.924	4.727
8	1.190	2.401	4.165	2.432	2.604	2.459	0.884	1.632	4.207
9	1.263	2.176	3.726	2.292	2.265	2.194	1.039	1.557	3.799
10	0.995	1.941	3.426	2.074	2.092	1.932	0.938	1.761	3.376
11	0.972	1.914	3.148	1.808	1.784	1.979	0.876	1.711	3.121
12	0.839	1.673	2.931	1.902	1.827	1.803	0.865	1.469	2.831
13	0.825	1.691	2.743	1.809	1.658	1.709	0.796	1.520	2.599
14	0.858	1.627	2.505	1.622	1.509	1.735	0.796	1.402	2.452
15	0.775	1.450	2.386	1.513	1.516	1.624	0.778	1.411	2.360
16	0.770	1.479	2.253	1.575	1.547	1.532	0.708	1.413	2.210

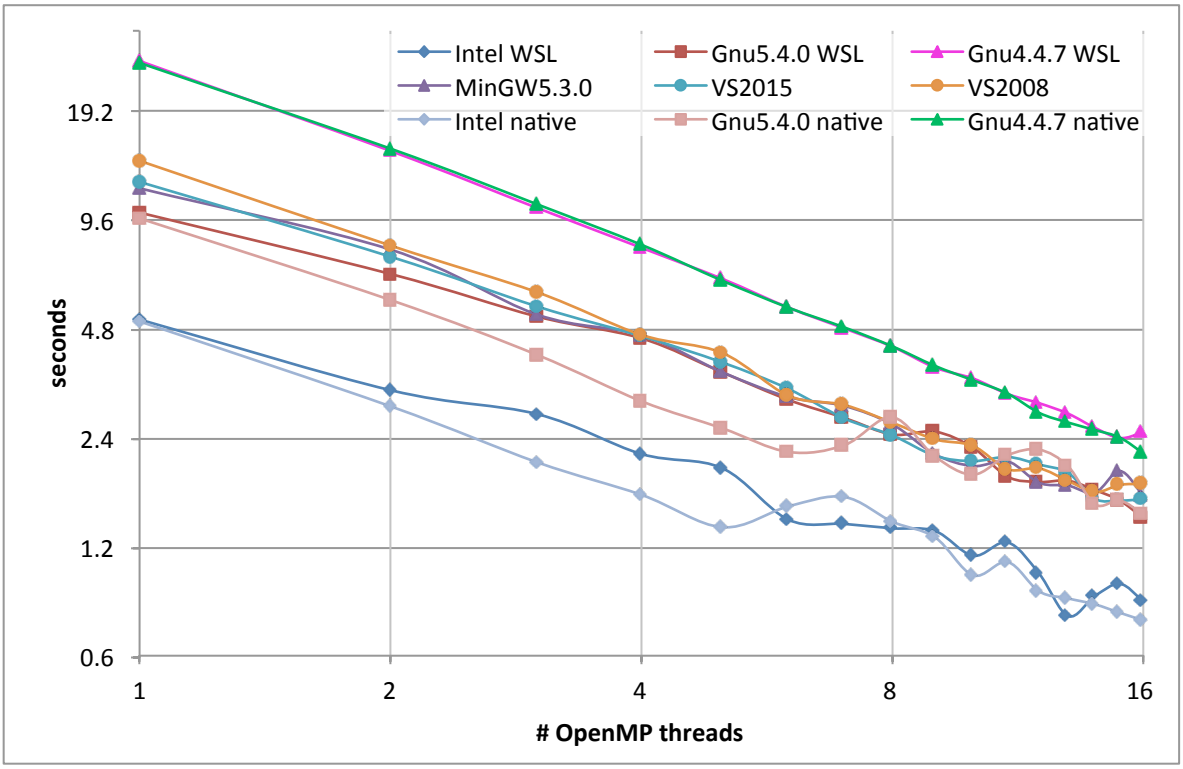


Figure 2: Graphs of execution times in seconds of the π calculation test job as a function of OpenMP threads for Phaser built with different compilers and running on different un-patched platforms.

Benchmarks of the π-program on patched platforms

Figure 2 and table 3 are the execution times for the same calculations as above but for the platforms patched for the Meltdown bug.

It is seen that the benchmark times on the patched platforms are roughly the same for all the Linux executables regardless of whether they are running on native Ubuntu or WSL. But all the Windows executables have increased their run time with up to 25% for single threaded use (VS2015: 12.223 seconds on the patched platform compared to 9.727 on the un-patched platform). On the other hand this discrepancy becomes smaller when the number of threads are increased.

Phaser

Phaser is a crystallographic software program making extensive use of the C++ Standard Template Library (STL) for manipulating dynamically allocated data objects that may often exceed hundreds of megabytes of memory. It also can take advantage of OpenMP parallelization. Much of the program runs sequentially however, as many functions cannot easily be parallelized. Given particular input data for a Phaser calculation which ultimately determines the size of dynamically allocated data objects for the calculation OpenMP parallelized loops may or may not significantly reduce the overall runtime of the calculation. Source details given in "Details of Phaser source versions" in the appendix.

The tests calculations discussed below were chosen with the criteria to be single component searches with clear unambiguous MR solutions to run for more than one but less than 20 minutes. As with the **π** program Phaser was compiled for maximum speed and with OpenMP for all compilers and as static executables, except for the Gnu 4.4.7 build

Table 3: Execution times in seconds of the π calculation test job as a function of OpenMP threads for Phaser built with different compilers and running on different patched platforms.

OpenMP threads	Intel WSL	Gnu5.4.0 WSL	Gnu4.4.7 WSL	MinGW 5.3.0	VS2015	VS2008	Intel native	Gnu5.4.0 native	Gnu4.4.7 native
1	5.093	10.062	26.355	11.719	12.223	13.973	5.034	9.697	26.016
2	3.253	6.811	14.904	7.954	7.604	8.164	2.936	5.774	15.122
3	2.794	5.198	10.379	5.266	5.545	6.070	2.059	4.076	10.622
4	2.171	4.528	8.066	4.625	4.595	4.634	1.675	3.039	8.220
5	1.984	3.651	6.625	3.656	3.890	4.134	1.364	2.561	6.549
6	1.432	3.068	5.524	3.125	3.290	3.153	1.557	2.204	5.521
7	1.394	2.737	4.840	2.953	2.728	2.972	1.653	2.298	4.872
8	1.355	2.461	4.308	2.625	2.449	2.656	1.413	2.745	4.319
9	1.332	2.507	3.774	2.172	2.152	2.389	1.280	2.140	3.814
10	1.139	2.263	3.521	2.016	2.081	2.292	1.006	1.908	3.481
11	1.239	1.883	3.191	2.078	2.136	1.976	1.095	2.160	3.200
12	1.017	1.822	3.011	1.813	2.036	1.988	0.908	2.235	2.832
13	0.777	1.836	2.818	1.781	1.931	1.835	0.869	2.012	2.663
14	0.882	1.730	2.576	1.703	1.638	1.710	0.837	1.581	2.540
15	0.953	1.619	2.415	1.953	1.620	1.791	0.793	1.619	2.420
16	0.854	1.449	2.499	1.672	1.633	1.801	0.756	1.484	2.192

where we used a Phenix installation of the Phenix-1.13-rc1-2965 version built on Centos6.

MR_AUTO on PDB entry 1ioM

This test consists of a data set with 86550 reflections. The keyword script is listed in figure 12 in the appendix. From figure 3 and the corresponding data in Table 4 it is apparent that for all executables and platforms there is no significant benefit from using OpenMP with more than about five or six threads.

Benchmarks of an MR_AUTO run with Phaser on 1ioM on un-patched platforms

In this calculation the executables built by the Microsoft compilers are the fastest if more than one OpenMP thread is used. Figure 4 shows the CPU usage as a function of wall time as recorded during the Phaser calculation with 16 OpenMP threads being allowed.

Considerable differences between the CPU loads of the executables can be seen. Although the calculation with the VS2015 executable is the fastest compared to the other executables it appears to spend more than 100 seconds within the first OpenMP parallelized loop whereas the Intel and the Gnu executables spend less than 50 seconds. On the other hand the VS2015 executable more than makes up for that sluggishness outside the parallelized loop since it finishes the calculation in less than 500 seconds.

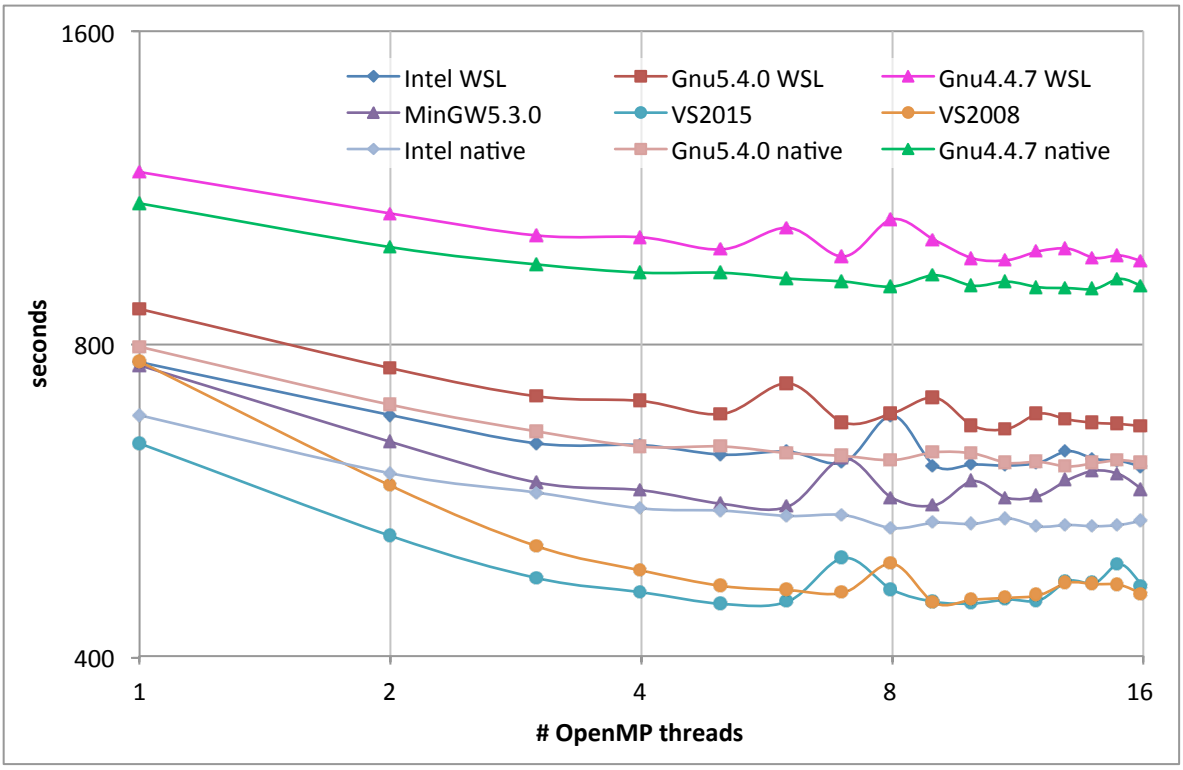


Figure 3: Graphs of execution times in seconds of the MR_AUTO on 1ioM test job as a function of OpenMP threads for Phaser built with different compilers and running on different un-patched platforms.

Table 4: Execution times in seconds of the MR_AUTO on 1ioM test job as a function of OpenMP threads for Phaser built with different compilers and running on different un-patched platforms.

OpenMP threads	Intel WSL	Gnu5.4.0 WSL	Gnu4.4.7 WSL	MinGW 5.3.0	VS2015	VS2008	Intel native	Gnu5.4.0 native	Gnu4.4.7 native
1	767.45	864.17	1170.32	762.39	641.45	768.3	682.07	793.82	1091.42
2	682.89	757.74	1067.72	643.62	522.5	584.24	600.16	698.72	990.89
3	641.36	712.01	1016.93	588.2	475.96	510.89	575.03	658.62	953.6
4	639.05	704.69	1012.92	578.32	461.35	484.34	555.38	636.97	936.61
5	625.68	684.83	986.27	561.17	449.6	468.04	552.64	637.17	936.31
6	629.86	732.58	1034.38	558.01	452.27	463.77	546.2	627.89	924.4
7	615.25	671.62	970.86	620.36	497.68	461.36	547.17	623.91	918.35
8	681.97	685.8	1054.11	568.42	463.83	491.66	531.66	618.24	908.26
9	610.35	709.89	1006.31	559.72	451.6	450.62	538.1	629.13	931.18
10	612.28	666.84	966.61	590.15	450.07	454.21	536.65	627.29	909.87
11	610.78	662.82	963.15	568.43	453.98	455.92	543.32	614.5	918.04
12	614.82	685.35	982.43	571.36	453.14	458.61	533.57	615.77	906.73
13	630.53	676.28	987.78	591.8	472.84	471.43	535.41	609.63	905.16
14	619.69	671.5	967.76	603.99	470.91	470.01	533.88	614.14	903.73
15	616.96	669.77	973.02	598.67	490.95	469.15	535.59	618.11	923.75
16	608.77	666.39	960.16	578.55	467.45	459.37	540.73	614.55	908.38

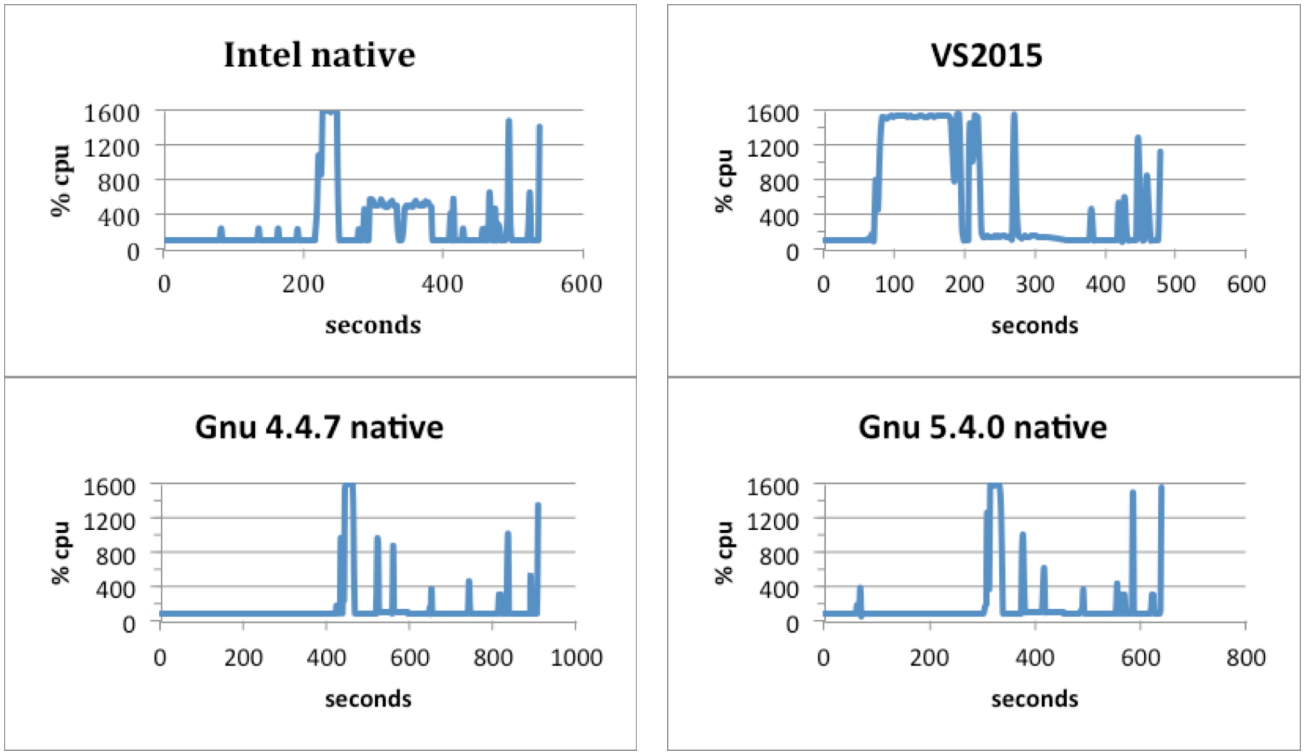


Figure 4: CPU usage graphs for four selected executables running on un-patched platforms with 16 OpenMP threads for the MR_AUTO on 1ioM test job.

Benchmarks of an MR_AUTO run with Phaser on 1ioM on patched platforms

Figure 5 and table 5 display the execution times for the same calculations but for the platforms patched for the Meltdown bug.

We note that the Windows executables suffer a run time increase of up to 37% (VS2015: one OpenMP thread 881.37 seconds on patched platform compared to 641.45 seconds on the un-patched platform). For the Linux executables there is virtually no change in speed when running them on patched or un-patched Ubuntu platforms. But running them on WSL they also suffer a performance hit of up to 25% increase in execution time.

Although not shown here, the graphs of the CPU usage as a function of wall time recorded during the Phaser calculation with 16 OpenMP threads being allowed are very similar to the graphs of executables running on un-patched platforms in figure 4.

MR_AUTO on 1HBZ

This test consists of a data set with 95355 reflections. The keyword script is listed in figure 13 in the appendix. In figure 6 it is apparent that for all executables and platforms there is no significant benefit from using OpenMP with more than about five or six threads. In fact on the WSL platform and for the win32 executables a slight overhead occurs if more than about 8 threads are used. This is not so for the Intel and Gnu executables running on native Ubuntu.

Benchmarks of an MR_AUTO run with Phaser on 1HBZ on un-patched platforms

For this test the Intel executable on native Ubuntu has the fastest execution time closely followed by the Gnu5.4.0 executable on native

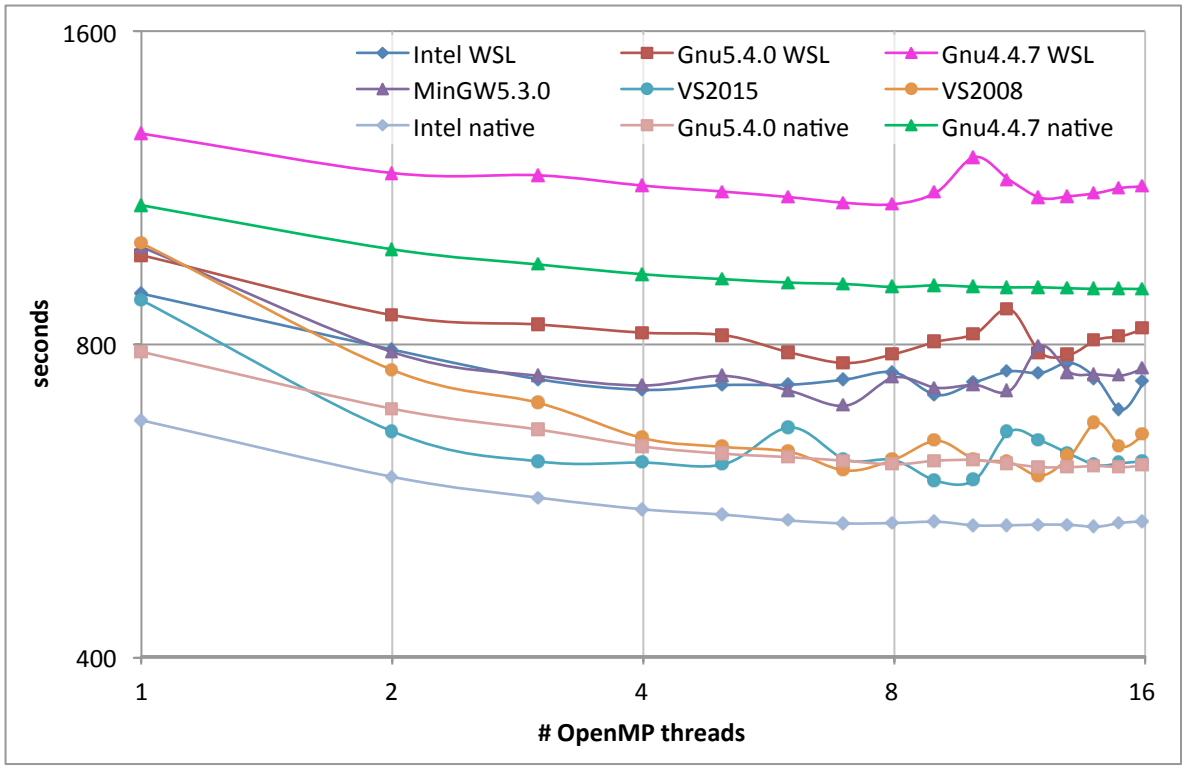


Figure 5: Graphs of execution times in seconds of the MR_AUTO on 1ioM test job as a function of OpenMP threads for Phaser built with different compilers and running on different patched platforms.

Table 5: Execution times in seconds of the MR_AUTO on 1ioM test job as a function of OpenMP threads for Phaser built with different compilers and running on different patched platforms.

OpenMP threads	Intel WSL	Gnu5.4.0 WSL	Gnu4.4.7 WSL	MinGW 5.3.0	VS2015	VS2008	Intel native	Gnu5.4.0 native	Gnu4.4.7 native
1	893.92	973.13	1274.63	991.73	881.37	999.68	674.85	785.34	1086.48
2	789.47	852.26	1167.63	785.51	658.87	755.02	595.36	692.44	985.95
3	739.27	834.61	1161.77	744.93	616.52	701.71	568.54	661.14	953.29
4	722.31	819.72	1135.5	729.48	614.98	649.24	554.11	636.93	933.04
5	729.94	815.17	1120.59	745.11	612.9	636.86	547.61	626.89	922.95
6	730.49	784.57	1106.87	720.79	664.67	629.89	540.58	622.05	915.99
7	739.17	766.55	1093.13	697.73	618.86	604.46	537.15	617.16	913.09
8	750.7	782.09	1090.04	742.59	618.31	619.45	537.67	612.31	907.21
9	713.89	803.67	1119.75	725.72	590.35	645.63	539.16	617.37	910.51
10	734.34	817.78	1209.7	730.37	592.19	619.68	534.82	618.25	907.56
11	752.3	863.43	1149.69	720.76	658.54	615.76	534.92	613.11	906.36
12	749.82	783.43	1105.23	795.82	645.31	597.56	535.74	608.59	906.52
13	766.81	781.07	1109.49	750.69	626.7	624.14	535.36	608.6	904.57
14	739.92	807.06	1117.08	747.56	611.49	672.13	533.23	610.11	903.43
15	691.24	813.66	1130.17	745.79	614.25	637.13	538.02	608.51	903.62
16	737.22	827.17	1133.72	757.98	617.17	655.36	539.52	610.9	902.94

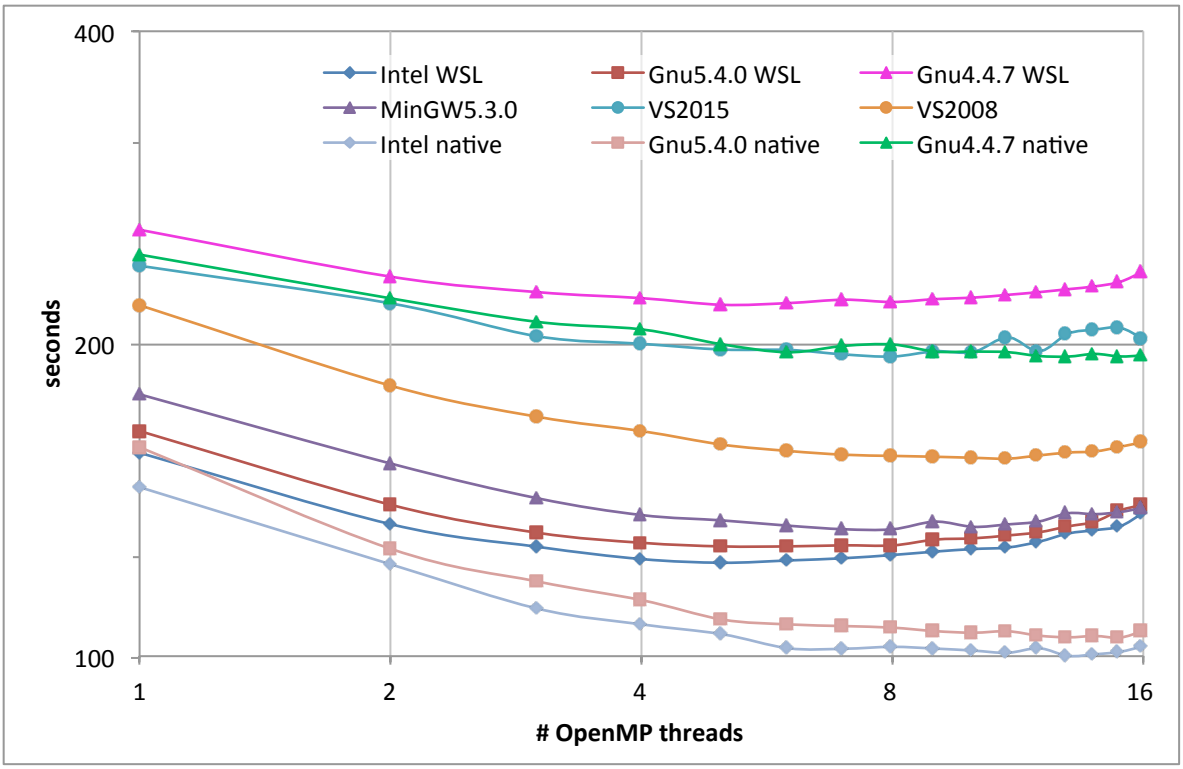


Figure 6: Graphs of execution times in seconds of the MR_AUTO on 1HBZ test job as a function of OpenMP threads for Phaser built with different compilers and running on different un-patched platforms.

Ubuntu as seen in table 6. Among the win32 executables it is interesting to note the ranking with the MingGW5.3.0 executable faster than the VS2008 executable that in turn is faster than the VS2015 executable. Using 16 OpenMP threads the Gnu4.4.7 executable runs almost twice as slowly as the Gnu 5.4.0 executable.

Benchmarks of an MR_AUTO run with Phaser on 1HBZ on patched platforms

For this test calculation the execution times on the patched platforms differ significantly from the un-patched runs. In figure 7 the most immediate observation is the significant overhead when running Linux executables on the WSL platform with more than 4 OpenMP threads. All three executables run slower with 16 threads than with just one thread. The execution times of the Windows executables plateau for more than five OpenMP threads.

The Linux executables running on native Ubuntu are practically unaffected by the patches for the Meltdown bug as seen in table 7. The increase in execution time is mostly under one percent regardless of number of OpenMP threads. This is contrary to the Windows executables where the fastest Windows executable, MinGW 5.40, suffer from an increase in execution time of between 14% and 24%.

MR_FRF on vz170x37_P1

This test consists of a data set with 156982 reflections. The keyword script is listed in figure 14 in the appendix. The fast rotation mode in Phaser predominantly exercises one of the OpenMP parallelized functions within Phaser. Avoiding serial single processing calculations should better inform us on how OpenMP execution times scales with the number of threads for Phaser.

Table 6: Execution times in seconds of the MR_AUTO on 1HBZ test job as a function of OpenMP threads for Phaser built with different compilers and running on different un-patched platforms.

OpenMP threads	Intel WSL	Gnu5.4.0 WSL	Gnu4.4.7 WSL	MinGW 5.3.0	VS2015	VS2008	Intel native	Gnu5.4.0 native	Gnu4.4.7 native
1	156.98	164.75	257.41	178.71	237.67	217.77	145.46	158.94	243.62
2	134.08	139.98	232.08	153.38	218.72	182.29	122.67	126.99	221.18
3	127.54	131.58	224.28	142.04	203.37	170.19	111.23	118.13	209.96
4	124.12	128.69	221.31	136.86	200.03	164.89	107.42	113.37	206.5
5	123.12	127.66	218.07	135.25	197.49	160.01	105.1	108.6	199.78
6	123.74	127.67	218.9	133.68	197.33	157.79	101.89	107.44	196.34
7	124.36	127.97	220.54	132.63	195.37	156.48	101.78	107.04	199.13
8	125.24	127.94	219.43	132.62	194.46	156.08	102.21	106.67	199.73
9	126.19	129.6	220.83	134.88	196.74	155.78	101.82	105.85	196.63
10	126.91	129.96	221.56	133.34	196.4	155.43	101.37	105.48	196.62
11	127.33	130.79	222.93	134.06	202.91	155.13	100.87	105.82	196.37
12	128.99	131.72	224.27	134.93	196.7	156.25	101.89	104.79	194.69
13	131.34	133.45	225.74	137.41	204.55	157.24	100.18	104.41	194.46
14	132.32	134.85	227.14	137.13	206.37	157.62	100.54	104.72	195.61
15	133.43	138.22	229.41	137.76	207.24	159.22	101.09	104.35	194.52
16	137.42	139.98	234.64	139.13	202.23	160.97	102.32	105.87	194.91

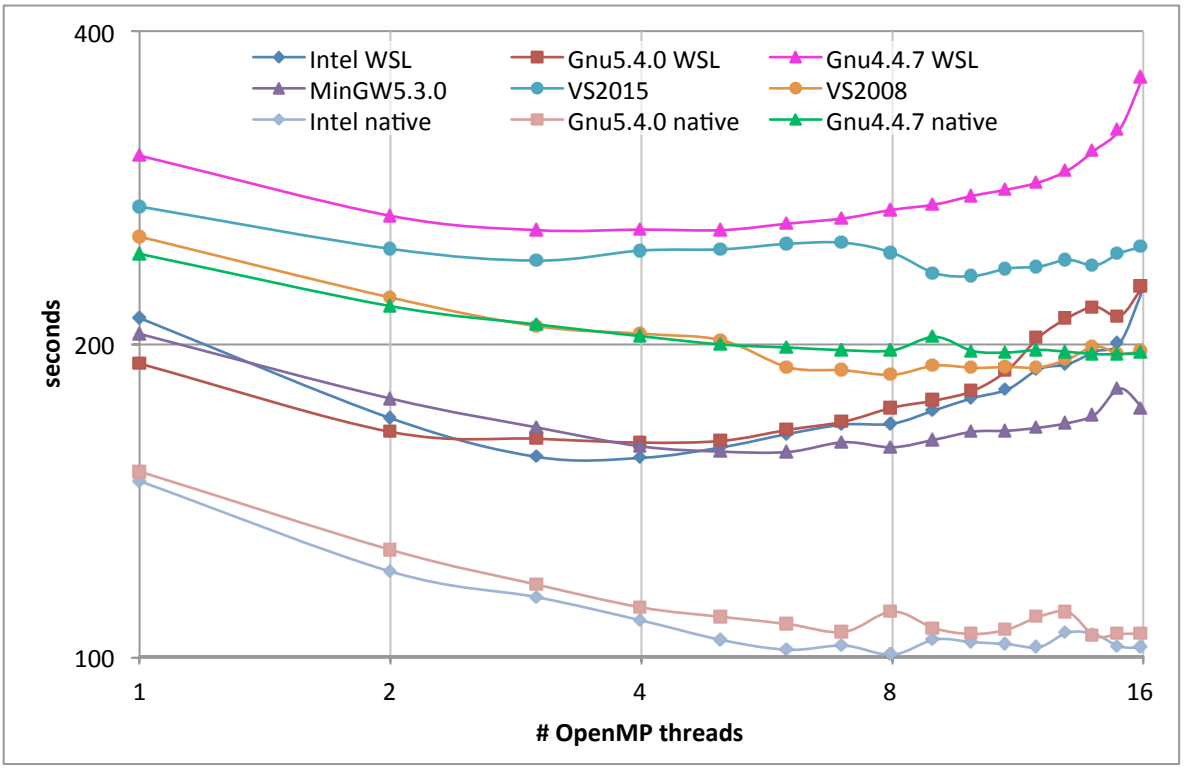


Figure 7: Graphs of execution times in seconds of the MR_AUTO on 1HBZ test job as a function of OpenMP threads for Phaser built with different compilers and running on different patched platforms.

Table 7: Execution times in seconds of the MR_AUTO on 1HBZ test job as a function of OpenMP threads for Phaser built with different compilers and running on different patched platforms.

OpenMP threads	Intel WSL	Gnu5.4.0 WSL	Gnu4.4.7 WSL	MinGW 5.3.0	VS2015	VS2008	Intel native	Gnu5.4.0 native	Gnu4.4.7 native
1	211.59	191.31	303.5	204.16	270.8	253.33	147.42	150.66	244.07
2	169.61	164.57	265.69	177.07	246.75	221.53	120.69	126.64	217.33
3	155.69	162.08	257.39	166.1	240.43	207.91	114.01	117.23	208.7
4	155.37	160.7	257.72	159.3	245.84	204.51	108.31	111.49	203.34
5	159.07	161.33	257.42	157.47	246.5	201.54	103.7	109.17	199.64
6	163.84	165.33	261.16	157.36	249.53	189.85	101.52	107.46	198.32
7	167.24	168.35	264.24	160.78	250.09	188.62	102.46	105.61	197.16
8	167.56	173.61	269.24	159	244.59	186.75	100.42	110.5	197.1
9	172.65	176.41	272.42	161.72	233.81	190.61	103.86	106.41	203.03
10	177.39	180.21	277.77	164.59	232.34	189.65	103.23	105.13	196.76
11	180.76	188.64	281.61	164.92	236.27	190	102.79	106.21	196.18
12	189.04	202.94	286.16	166.06	237.27	189.86	102.12	109.25	197.29
13	191.18	211.81	293.87	167.76	240.83	193.32	105.48	110.33	196.44
14	196.52	216.92	307.18	170.94	237.9	198.83	105.04	104.68	195.47
15	200.3	212.59	321.86	181	244.4	195.66	102.25	105.16	195.47
16	225.8	227.12	361.56	173.2	248.1	196.84	102.04	105.15	195.97

Benchmarks of an MR_FRF run with Phaser on vz170x37_P1 on un-patched platforms

Figure 8 shows the Intel executable on native Linux being the fastest closely followed by the Gnu5.4.0 executable on native Linux. With 13 OpenMP threads it executes the calculation in 29.81 seconds closely followed by the speed of the Gnu5.4.0 executable also on native Linux. This is about three times faster than the builds made with the Microsoft compilers. For a single threaded calculation the speed differences become less dramatic with the Intel or Gnu5.4.0 executable only being about 1.5 times faster than VS2015 executable. The slowest executable for this calculation is the VS2008 executable which only after using more than 8 OpenMP threads begin to catch up where the speed of other executables reach a plateau.

The CPU usage graphs on figure 9 of the executables with 16 OpenMP threads indicate a peculiar behavior. The Intel executable on native Ubuntu never quite attains full usage of the CPU but nevertheless finishes in 31 seconds. Both the Windows executables (MinGW 5.3.0 and VS2015) attain the full CPU usage of 1600% but apparently still run more than twice as slow as the Intel executable on native Ubuntu. On WSL however the Intel executable again never reaches 1600% usage but settles for about 500% and finishes the calculation in a similar time as the MinGW5.3.0 and the VS2015 executable. This points to inefficiencies in the WSL platform compared to native Linux.

Benchmarks of an MR_FRF run with Phaser on vz170x37_P1 on patched platforms

We note in figure 10 that the WSL platform is clearly affected for this calculation; the execution times plateau already above two OpenMP threads.

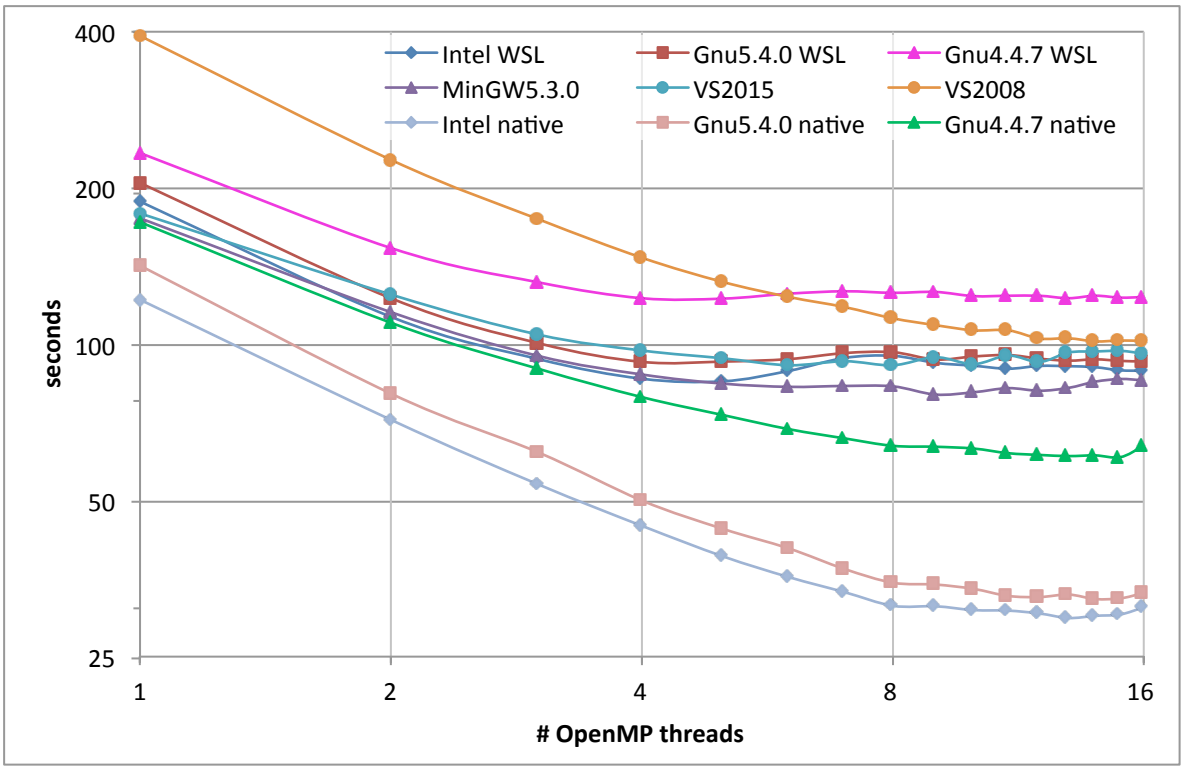


Figure 8: Graphs of execution times in seconds of the MR_FRF on vz170x37_P1 test job as a function of OpenMP threads for Phaser built with different compilers and running on different platforms.

Table 8: Execution times in seconds of the MR_FRF on vz170x37_P1 test job as a function of OpenMP threads for Phaser built with different compilers and running on different un-patched platforms.

OpenMP threads	Intel WSL	Gnu5.4.0 WSL	Gnu4.4.7 WSL	MinGW 5.3.0	VS2015	VS2008	Intel native	Gnu5.4.0 native	Gnu4.4.7 native
1	188.16	204.36	233.31	174.72	178.51	392.27	121.73	141.6	171.71
2	113.01	122.72	153.28	115.33	124.69	226.72	71.58	80.41	110.05
3	93.82	100.68	131.79	95.11	104.5	174.61	53.86	62.08	89.87
4	85.96	92.64	122.79	87.57	97.37	147.11	44.8	50.12	79.22
5	84.98	92.76	122.57	84.11	94.08	132.19	39.16	44.24	73.21
6	88.99	93.78	125.29	82.91	91.24	123.7	35.72	40.57	68.77
7	94.24	96.32	126.62	83.24	92.86	118.29	33.42	37.04	66.02
8	95.13	96.69	125.75	83.1	91.23	112.55	31.46	34.86	63.88
9	92.21	93.59	126.25	80.13	94.51	109.2	31.4	34.58	63.62
10	91.05	94.87	123.93	80.95	91.68	106.73	30.87	33.92	63.12
11	89.85	95.49	124.14	82.43	95.21	106.74	30.8	32.88	61.83
12	90.98	93.95	124.13	81.49	92.59	102.92	30.41	32.72	61.36
13	90.86	93.13	122.79	82.54	96.51	103.04	29.81	33.08	61.05
14	90.45	93.69	124.16	84.85	96.95	101.63	30.1	32.43	61.23
15	89.21	93.04	123.1	85.84	97.22	102.04	30.28	32.53	60.76
16	89.07	92.77	123.49	85.35	96.04	101.82	31.27	33.3	63.83

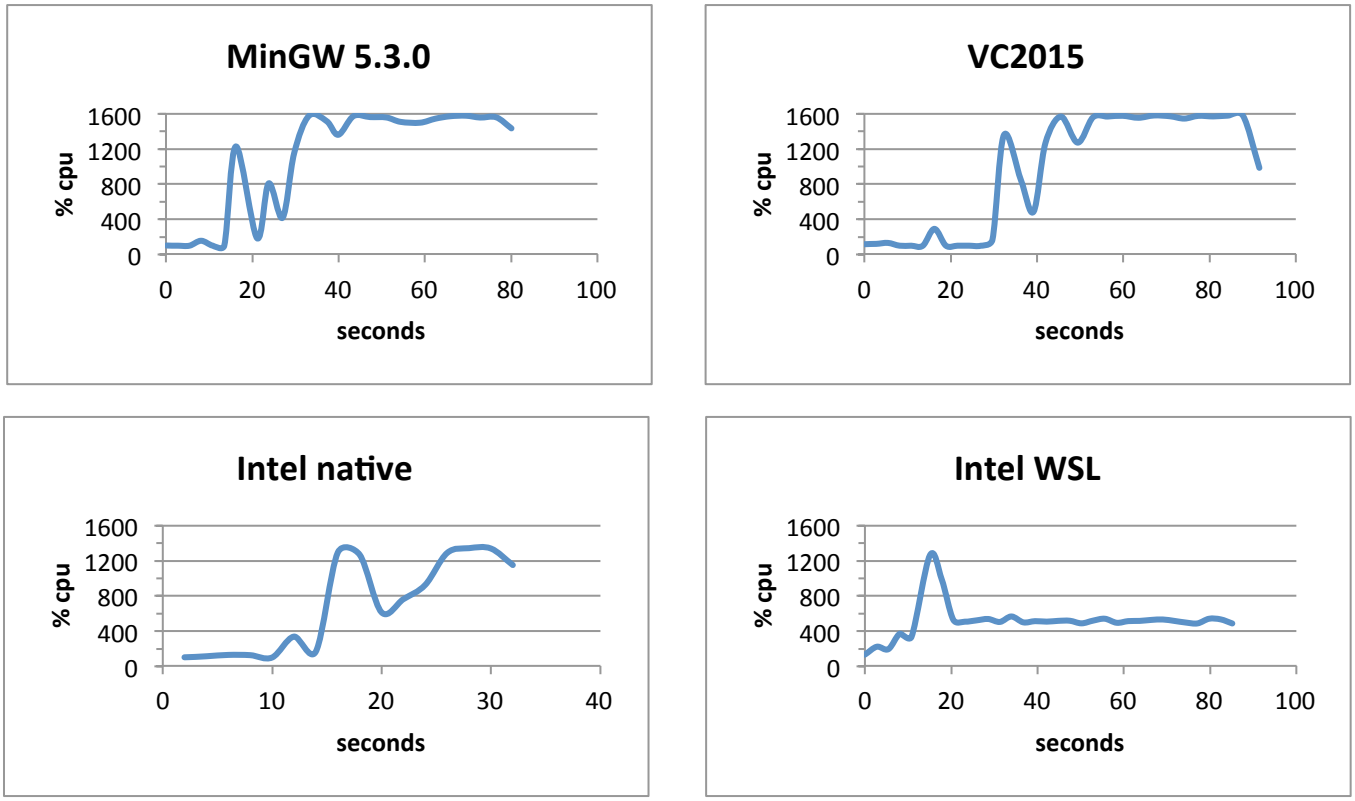


Figure 9: CPU usage graphs of four selected executables running on un-patched platforms with 16 OpenMP threads for the MR_FRF on vz170x37_P1 test job

From table 9 it is noted that the Linux executables running on native Ubuntu suffer from almost no performance hit after the platform has been patched for the Meltdown bug. The Intel executable time increases at most 3.5% when run with 16 threads and the Gnu 5.4.0 executable with at most 2.2%. The execution time of the Windows executables on the other hand increases dramatically with up to 60% in one instance for the VS2015 executable.

Although not shown, the graphs of the execution times for the tests on the patched platforms are quite similar to the graphs in figure 9.

Discussion

Drawing conclusions from the benchmark tests is greatly complicated by the unanticipated dependence on the type of calculations performed. It was thought the relative speed of an executable built with one compiler would remain the same when compared to an executable built with a different compiler regardless of the type of calculations. The test performed here demonstrates that this is not true. Nevertheless there are several points to note from the benchmark tests.

For small programs like the π-calculator the Intel compiler clearly comes out as a favourite. It produces executables that are three or four times as fast as executables built by the Gnu4.4.7 compiler and about twice as fast as executables built by the Gnu5.4.0 compiler. It is also noted that the execution speeds of all executables scale well with the

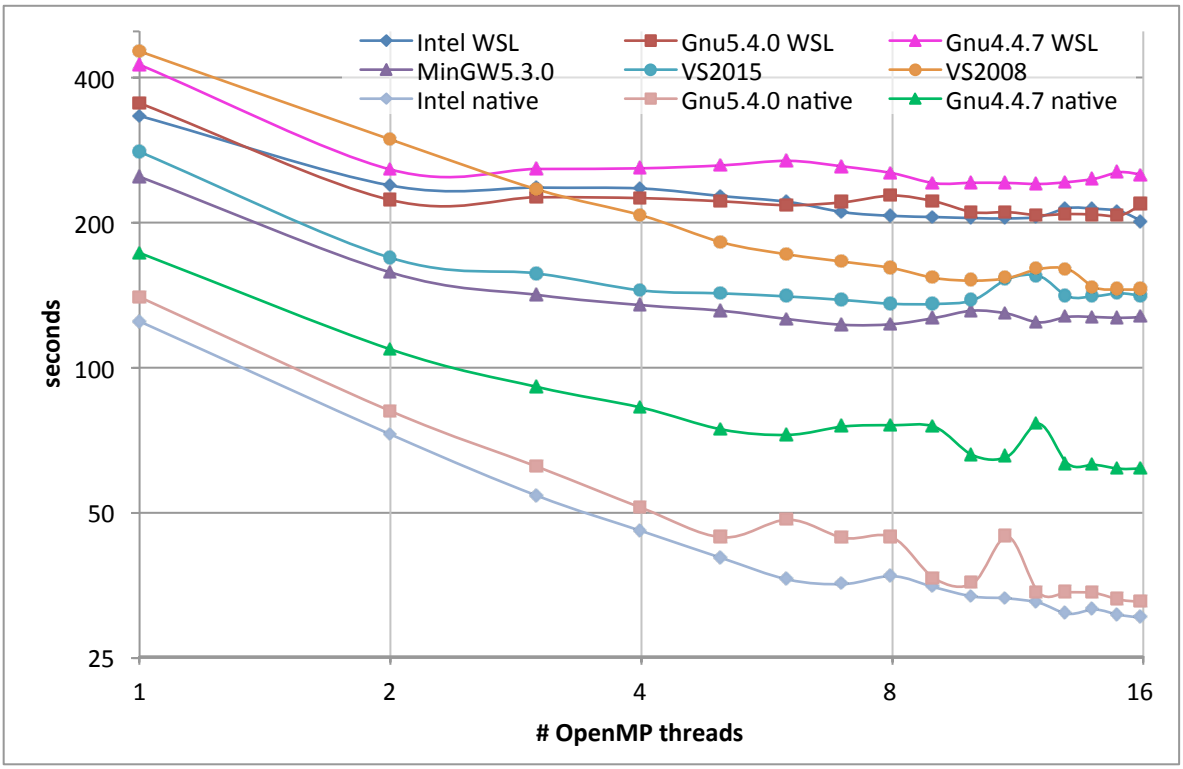


Figure 10: Graphs of execution times in seconds of the MR_FRF on vz170x37_P1 test job as a function of OpenMP threads for Phaser built with different compilers and running on different patched platforms.

Table 9: Execution times in seconds of the MR_FRF on vz170x37_P1 test job as a function of OpenMP threads for Phaser built with different compilers and running on different patched platforms.

OpenMP threads	Intel WSL	Gnu5.4.0 WSL	Gnu4.4.7 WSL	MinGW 5.3.0	VS2015	VS2008	Intel native	Gnu5.4.0 native	Gnu4.4.7 native
1	333.55	353.84	425.33	249.3	280.3	453.65	124.65	139.72	172.81
2	238.89	222.97	258.02	157.55	169.03	297.84	72.52	81.11	108.99
3	236.5	225.86	258.67	141.51	156.8	234.59	54.09	62.17	91.06
4	235.47	224.92	259.52	134.76	144.65	207.29	45.72	51.09	82.51
5	227.01	221.5	263.2	131.17	142.61	182.17	40.15	44.38	74.3
6	220.85	217.42	268.95	126.03	140.5	171.99	36.25	48.28	72.44
7	210.17	220.66	261.65	122.65	138.13	166.1	35.49	44.25	75.42
8	206.61	227.72	253.52	123.08	135.68	161.1	36.86	44.38	75.84
9	205.46	221.28	241.59	126.82	135.59	153.66	34.92	36.4	75.2
10	204.44	210.27	242.22	131.3	138.08	151.81	33.43	35.68	65.68
11	204.09	210.38	241.99	129.36	152.7	153.65	33.14	44.53	65.39
12	205.75	207.29	240.77	124.12	154.97	160.55	32.44	34.02	76.33
13	214.48	208.53	243.04	127.45	140.51	160.23	30.79	34.14	63.04
14	213.74	207.91	246.98	127.35	140.71	146.92	31.43	33.99	62.78
15	211.19	207.01	254.77	126.83	142.67	145.49	30.6	32.94	61.61
16	200.89	218.85	251.54	127.37	140.77	145.64	30.23	32.57	61.72

number of OpenMP threads. This suggests that the small size of the involved functions doing the calculations corresponds to an equally small number of assembler instructions that easily fit into cache memory on the CPU during execution.

For a large program like Phaser the picture is much more mixed; Phaser built with the Gnu 5.4.0 compiler for the calculations presented here, is almost as fast as Phaser built with the Intel compiler. As the Intel compiler uses the same C++ Standard Template Library as the Gnu compiler we speculate that because this library is used by both Phaser executables that could explain the lack of time differences.

Phaser built with the Gnu 4.4.7 compiler is generally the slowest in all tests except for the test of MR_FRF calculations where it remains faster or on par with Phaser executables for Windows regardless of the number of OpenMP threads.

Phaser built with the VS2015 compiler is generally faster than when built with VS2008, often more so when using just one OpenMP thread. The exception is the MR_AUTO on 1HBZ calculation where Phaser built with the VS2008 compiler happens to be somewhat faster than when built with the VS2015 compiler.

Back in 2008 benchmark tests using Phaser in MR_FRF mode were done on the vz170x37_P1 data set identical to the ones presented here. Although the tests back then were done on slower hardware they ran faster because the current Phaser version is doing more preliminary calculations before it begins the actual MR_FRF calculation. The conclusion back then was that a VS2005 executable on Windows XP was faster than g++ 4.1 executable on Fedora 7 when running Phaser with one OpenMP thread (329 seconds against 431 seconds). In light of our current results and assuming that the VS2008 compiler produces executables with the same speed or better than the VS2005 compiler this suggests that Gnu compilers have improved more quickly over the past ten years than have Microsoft compilers.

As for Phaser built with the MinGW 5.4.0 compiler its execution times are faster than when built with the Microsoft compilers except for the MR_AUTO on 1ioM calculation. In other words it is a competitive alternative to Microsoft compilers when building for the Windows platform.

Regarding OpenMP the tests carried out here indicates that OpenMP threading on native Linux scales better than on Windows 10. This is likely to be specific to the operating systems rather than differences between Windows compilers and Linux compilers in how they implement OpenMP threading.

Conclusion

From these tests one can conclude that it is worth using the more recent Gnu5.4.0 compiler when building for Linux. The Gnu4.4.7 compiler that is currently used produces executables often being half the speed of executables built with the Gnu5.4.0 compiler currently available in Ubuntu 16.04.

The striking advantage in the case of the Intel compiler for Linux builds of the small π-program does not hold for Phaser and presumable other large programs which use the C++ Standard Template Library. Phaser run with about the same speed when built with the Gnu5.4.0 compiler.

The Meltdown bug

Until the arrival of the patches for the meltdown bug deciding between platforms Linux and WSL the picture was clear. Running calculations on native Linux can be expected to run faster if the program is large and to run with the same speed on WSL if the program is small. Between Linux executables and win32 executables however there seemed to be no clear winner in terms of speed. Some calculations are faster on Windows and others are faster on Linux. Hence other priorities may determine the choice of platform.

However, the patches for the meltdown bug changes this conclusion in favour of native Linux where the execution times of Phaser is practically unaffected by the new patches. In contrast the Windows 10 platform is affected far more severely. It goes beyond the scope of this article to speculate what the exact source of these inefficiencies is. Moreover, it can be expected that the patches for the operating systems in time will improve and reduce the apparent inefficiencies currently experienced.

APPENDIX

Patches

Verification that the BIOS and the OS patches have been properly installed was done with the scripts available on https://github.com/speed47/spectre-meltdown-checker for Linux and https://support.microsoft.com/en-gb/help/4073119/protect-against-speculative-execution-side-channel-vulnerabilities-in for Windows.

π-program source

Program code is shown in figure 11.

Details of Phaser source versions

We used Phaser available from https://git.uis.cam.ac.uk/x/cimr-phaser/phaser.git (git hash: 731557) compiled with the CCTBX available from https://github.com/cctbx/cctbx_project (git hash: cb94e1b and 1f0b0593) as present on November 21, 2017. These versions are present in the Phenix-1.13-rc1-2965 build and the CCTBX-installer-dev-1230 build (no differences in C++ sources between revisions cb94e1b and 1f0b0593).

Keyword Scripts

In the keyword scripts below that were run from a Bash shell $ncpu is the parameter that specifies the number of OpenMP threads available throughout the execution. Figures 12–14 are mentioned in the main text.

```
// OpenMPtest.cpp an example of multithreading using OpenMP
// Compiler command line for
// Linux g++: "g++ -fopenmp -static -g -O3 OpenMPtest.cpp -oOpenMPtest.gnu.X"
// Linux Intel: "icpc -qopenmp -fast -static -debug all OpenMPtest.cpp -oOpenMPtest.intel.X"
// Windows Visual Studio: "cl OpenMPtest.cpp /Ox /fp:fast /EHsc /openmp /FeOpenMPtest.exe"

#define _USE_MATH_DEFINES
#include <omp.h>
#include <cmath>
#include <iostream>
#include <iomanip>

using namespace std;

class PiCalculator
{
  long lmax; // Gaussian integration slices
  long kmax; // terms in Pi series
  double gaussint;
  double D, b, elapsed_time;
  double GetSlowPi(long m);
public:
  PiCalculator(int nthreads);
  ~PiCalculator();
  void GaussIntegrate();
  double GetPiResult() { return gaussint; }
  double GetElapsedTime() { return elapsed_time; }
};

PiCalculator::PiCalculator(int nthreads)
{
  lmax = 20000; // Gaussian integration slices
  kmax = 30000; // terms in Pi series
  gaussint = 0.0;
  D = 20.0, b = 3.0;
  elapsed_time = 0.0;

  cout << "This program is the slow Pi calculator, a small OpenMP test for your system." << endl;
  const int nprocs = omp_get_num_procs();
  if (nprocs < nthreads)
  {
    cout << "\nWarning!\nRunning this calculation with more threads than processors makes no sense.\n"
      << "Unless you want to stress test your system don't use more than " << nprocs << " threads." << endl;
    nthreads = nprocs;
  }
  omp_set_num_threads(nthreads);
}

PiCalculator::~PiCalculator()
{
  cout << "\nThanks for using the slow Pi calculator." << endl;
}

double PiCalculator::GetSlowPi(long m)
{// Compute Pi = 3*sqrt(3)*Sum_{k=1}(-1)^k/(3k+1) - log(2)*sqrt(3)
 // hence converges fairly slowly
  double pisum = 0.0;
  for (long k = 0; k<(kmax + m); k++)
  {
    double frac, div = 3.0*k + 1.0;
    if (k % 2 == 0) // i.e. k is even
      frac = 1.0 / div;
    else
      frac = -1.0 / div;
    pisum += 3.0*sqrt(3.0)*frac;
  }
  pisum -= log(2.0)*sqrt(3.0);
  return pisum;
}
```

Figure 11: Source code for the π-program

```
void PiCalculator::GaussIntegrate()
{
  double start, finish;
  start = omp_get_wtime();
  double sum = 0.0;
  const double dx = 2.0*D / lmax;
#pragma omp parallel
  {
#pragma omp single
    cout << "Number of threads is: " << omp_get_num_threads() << endl;
#pragma omp for reduction(+ : sum) // add all openmp for-loop results into the variable gaussint
    for (long m = 0; m< lmax; m++)
    {
      // integrate a Gaussian multiplied with Pi
      double x = m*dx - D;
      sum += GetSlowPi(m)
        *1.0 / sqrt(2.0*b*GetSlowPi(m + 7))*exp((cos(GetSlowPi(m + 13))*x*x) / (2.0*b)) * dx;
    }
  }
  // end #pragma omp parallel
  finish = omp_get_wtime();
  gaussint = sum;
  elapsed_time = (finish - start);
}

int main()
{
  int nthreads = 1;
  cout << "Enter number of OpenMP threads to use for this calculation: ";
  cin >> nthreads;
  PiCalculator myPi(nthreads);
  cout << setprecision(18);
  myPi.GaussIntegrate();
  cout << "\rtime= " << myPi.GetElapsedTime() << " sec, gausssum = " << myPi.GetPiResult() << endl;
  return 0;
}
```

Figure 11: Source code for the π-program (cont.)

```
HKLIN ../1ioM.mtz
LABIN F = FOBS_X  SIGF = SIGFOBS_X
COMPOSITION PROTEIN SEQUENCE ../1ioM_ChainA.seq NUM 1
MODE MR_AUTO
ENSEMBLE MR_2R26_A PDB ../sculpt_2R26_A_singlechain.pdb  IDENTITY 34.218
SEARCH ENSEMBLE MR_2R26_A NUM 1
JOBS $ncpu
```

Figure 12: Keyword script for running Phaser with the data set with the PDB code 1ioM. The model file, `sculpt_2R26_A_singlechain.pdb`, is derived from chain A of the structure with PDB code 2R26 and has been trimmed with the program Sculptor.

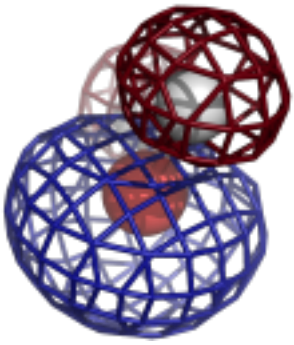
```
HKLIN ../1hbz.mtz
LABIN F = FOBS_X  SIGF = SIGFOBS_X
COMPOSITION PROTEIN SEQUENCE ../1HBZ_ChainA.seq NUM 1
MODE MR_AUTO
ENSEMBLE MR_1A4E_A PDB ../sculpt_1A4E_A.pdb  IDENTITY 41.393
SEARCH ENSEMBLE MR_1A4E_A NUM 1
JOBS $ncpu
```

Figure 13: Keyword script for running Phaser with the data set with the PDB code. The model file, `sculpt_1A4E_A.pdb`, is derived from chain A of the structure with the PDB code 1HZB and has been trimmed with the program Sculptor.

```
HKLIN ../vz170x37_P1.mtz
MODE MR_FRF
LABIN F=F_vz SIGF=SIGF_vz
ENSEMBLE octamer PDBFILE ../octamer.pdb IDENT 1.0
COMPOSITION PROTEIN MW 18000 NUMBER 32
SEARCH ENSEMBLE octamer NUM 1
MACANO PROTOCOL OFF
MACTNCS PROTOCOL OFF
RESOLUTION HIGH 0.5
RESOLUTION AUTO HIGH 0.5
JOBS $ncpu
```

Figure 14: Keyword script for running Phaser with the VZ170x37_P1 data set. This is in-house data kindly provided by Andrea Mattevi. The model file, octamer.pdb, consists of 8 NCS copies of chain A of the hexameric structure with the PDB code 2W5E sited in the corners of a twisted cube.

VOLUME NINE



COMPUTATIONAL CRYSTALLOGRAPHY NEWSLETTER

JULY MMXVIII

1.14, CABLAM, VICINAL SS

Table of Contents

- Phenix News 45
- Expert Advice
  - Fitting tips #16 – Vicinal disulfides: Have you seen one of these strange gems? 46
- Short Communications
  - CaBLAM: A C-Alpha Based Low-resolution Annotation Method for secondary structure and validation 51
  - Tools for interpreting cryo-EM maps using models from the PDB 58
- Articles
  - Using the New Program Template 62

Editor

Nigel W. Moriarty, NWMoriarty@LBL.Gov

Phenix News

Announcements

Phenix 1.14 release

The Phenix developers are pleased to announce that version 1.14 of Phenix is now available (build 1.14-3260). Binary installers for Linux, Mac OSX, and Windows platforms are available at the download site.[1] Highlights for this version include new tools and feature enhancements:

Reorganization, updates and addition of cryo-EM tools

- phenix.mtriage - assess map and model quality
- phenix.auto_sharpen - map sharpening
- phenix.map_symmetry - identify symmetry in maps
- phenix.map_box - cut out unique parts of maps
- phenix.combine_focused_maps - combine different maps
- phenix.dock_in_map - automatically place an atomic model into a map
- phenix.map_to_model - automatically build atomic model from a map
- phenix.sequence_from_map - identify sequence from a map
- phenix.real_space_refine - improved refinement of models
- phenix.validation_cryoem - separate tool for comprehensive validation of models and maps
- phenix.cablam_idealization - Tool to automatically fix Cablam outlier

eLBOW

- better support for metals and metal clusters
- added plugin for QM package Orca

[1] http://phenix-online.org/download/

Phaser-2.8.2

- bugfixes
- Phassade substructure search when starting from seed substructure
- fix crash in MR_ATOM
- problems with cumulative intensity distribution for extremely weak data
- improve computation and presentation of data information content

Performance improvements

- NCS search
- Generation of secondary structure restraints
- Clashscore calculation
- AmberPrep is more robust
- Restraints for ARG improved

New Phenix video tutorials

GUI

- New section in main window for cryo-EM tools
- Separate validation GUI for cryo-EM structures
  - Added phenix.map_symmetry
  - Added phenix.dock_in_map
  - Added phenix.map_to_structure_factors
  - Added phenix.combine_focused_maps
  - Added phenix.sequence_from_map

phenix.ligand_identification:

- Added an option to generate ligand library based on sequence and structural homologs of the input pdb model.

Amber

A new command, amber.h_bond_information, has been added to use the Amber H-bond detection code. Once AmberTools is installed, the command will provide the total number of H-bonds into a protonated model.

Expert advice

Fitting Tip #16 – Vicinal disulfides: Have you seen one of these strange gems?

Jane Richardson and Lizbeth Videau

Duke University

What is a vicinal SS?

A vicinal SS is a disulfide between two sequence-adjacent cysteine residues, with the rather startling appearance shown in figure 1 for a 1.75Å-resolution example in 1wd3. Early calculations implied that a vicinal SS could form only with a *cis* peptide (Chandrasekharan 1969) with the first two protein-crystal examples were both fit as *cis*, although undeposited. When both of those cases were shown to actually be *trans*, most studies including the one review (Carugo

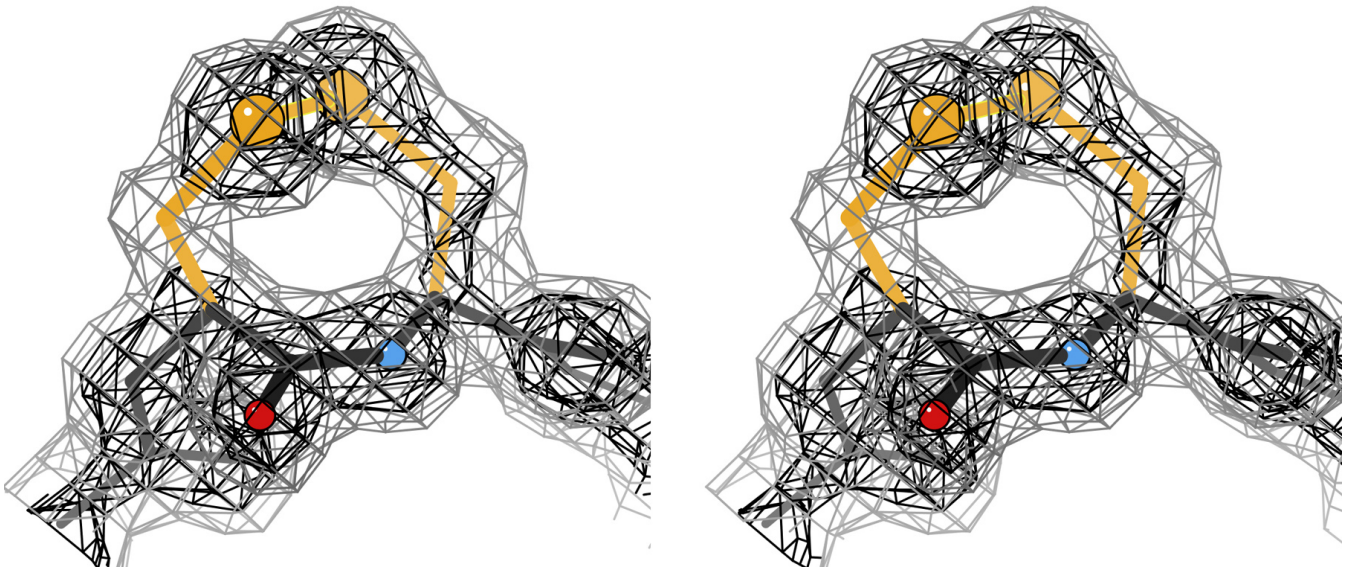


Figure 1: Stereo of a *cis* vicinal disulfide in C-conformation, in its clear $2F_o$-F_c electron density at 1.75Å (1.2σ contours in gray, 3σ in black) Cys 177 from the 1wd3 arabino-furanosidase (Miyanaga 2004).

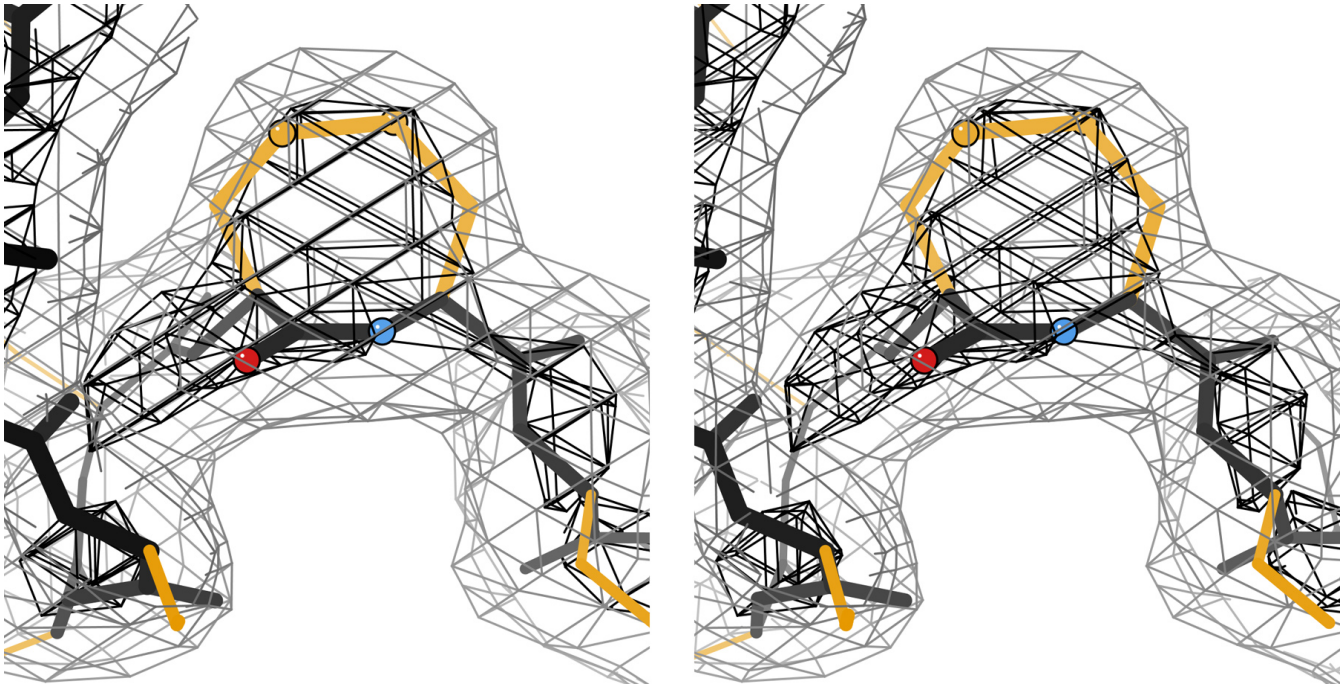


Figure 2: Stereo of a *cis* vicinal disulfide, also in C- conformation, shown as bonded by the electron density at 3.1Å resolution. Cys e 28 from the 4tvp human anti-HIV Fab in complex with env (Pancera 2014). The vicinal SS is in a variable loop of the light chain, near but not at the binding site.

2003) assumed vicinal SS would all be *trans.* It is by now indisputable that the intervening peptide can be either *cis* or *trans*, with *trans* more common – however, both forms are quite rare. Vicinal SS occurrence patterns, conformations and functions were recently reviewed (Richardson 2017) based on quality-filtered reference data and hand-curation of examples.

Most sequence-adjacent Cys residues (not bridged) are in the reduced SH form and only rarely can both Cys ligand the same metal. If they are in the oxidized state, they usually bond to different partners rather than to each other, especially in small SS-rich proteins. Vicinal SS are the exception to all the above rules. Like most rare and energetically unfavorable arrangements, when they do genuinely occur, they are almost always functionally important and thus well worth examining. There are surprisingly few vicinal SS with redox functionality, but they confer stability, bind ligands and gate large conformational changes.

Appearance at various resolutions

At high to medium resolution a vicinal disulfide is very clear and obvious unless there is substantial local disorder. In well-ordered regions, even at 3Å, they can be recognized as a large, dense protrusion from the backbone (see figure 2). However, the evidence can disappear near 4Å (discussed later) or in partially disordered regions. Assigning the detailed conformation is more difficult, of course, since that depends on positions of the carbonyl O and Cβ atoms, which disappear somewhere between 2.5 and 3Å resolution. Even so, trying the four possibilities shown in the next section might suggest a preferred answer. Keep in mind that a vicinal SS is somewhat strained and susceptible to radiation damage, so if the bond is seen to be partially or fully broken, that does not prove it was not originally SS-

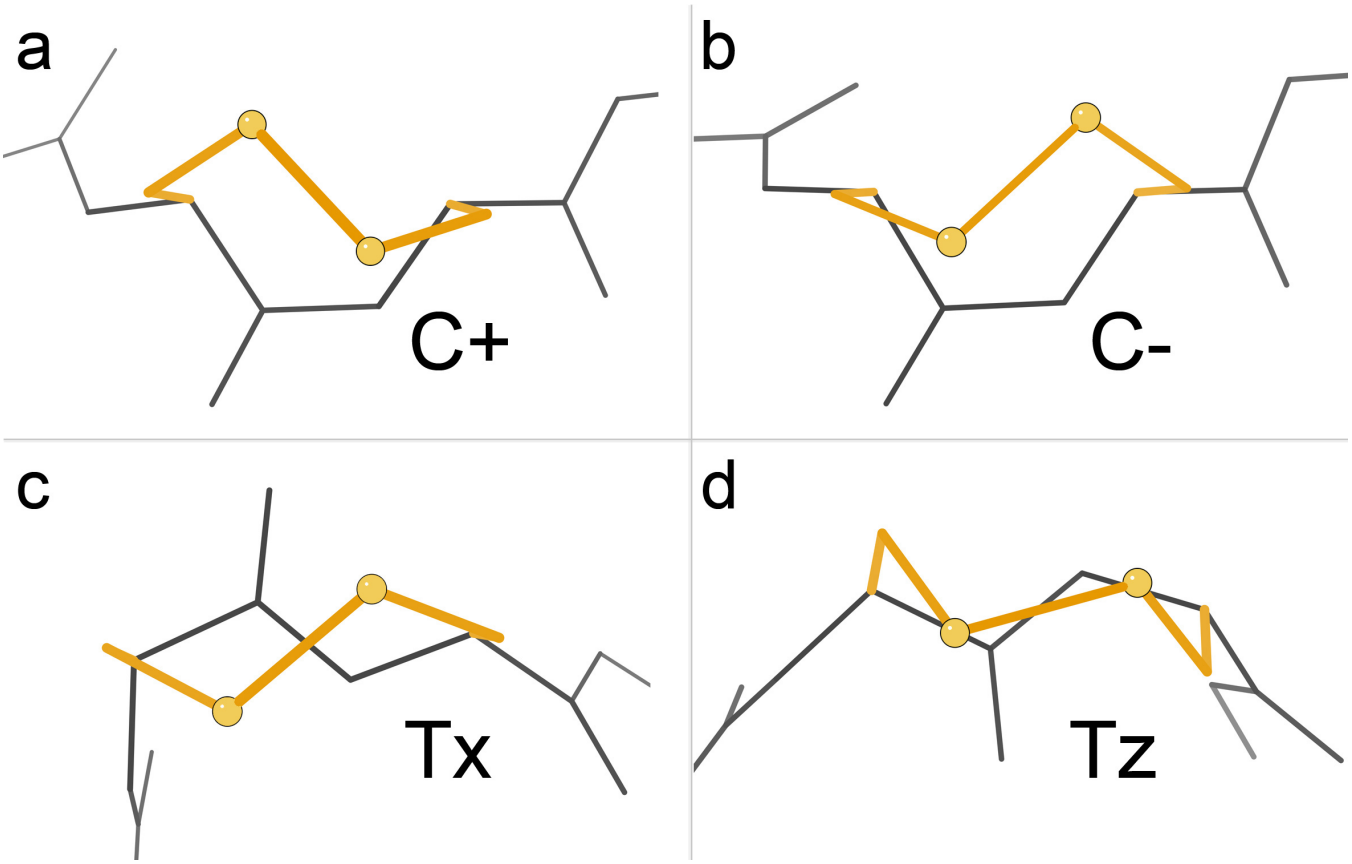


Figure 3: The four conformations of vicinal disulfides seen in reliable examples. a) *Cis* peptide with righthanded χ3, called C+. b) *Cis* with lefthanded χ3, called C-. c) *Trans* and lefthanded, with the SS making an X shape relative to the peptide bond, called Tx. d) *Trans* and lefthanded, with the SS making a Z shape relative to the peptide, called Tz.

bonded (see 3t4m, with Cys-*trans*-Cys191 bonded in chain B, broken in chain E, and probably a mixture in chain D).

Conformations

Only four different conformations are observed in reliable examples, two for *cis* and two for *trans*, as compared in figure 3. For *cis*, disulfide handedness changes between the two conformations, and for *trans*, the peptide orientation changes. The names of the conformations change sign (C+ vs C-) or describe the shape as viewed from above (Tx vs Tz). No cases of dynamic interconversion between *cis* and *trans* are seen, not surprising given the tight, rather strained ring. Specifications of these conformations for use in model building are given in Richardson 2017. These four conformations also match the four distinct cases observed by NMR for dipeptides in solution (Creighton 2001).

Functions

The best-known vicinal disulfide is the Cys-*trans*-Cys of the "C loop" of α subunits in the pentameric nicotinic acetylcholine receptor (nAchR). The vicinal SS is essential both for agonist binding and for the large conformational change that couples that binding to ion channel opening. Figure 4 shows it for the closed, agonist-binding state. In this cryoEM map at 3.7Å one can see it is positioned to touch the ligand, but confirmation of the actual SS bond comes from decades of biochemical and genetic analysis and from higher-resolution structures.

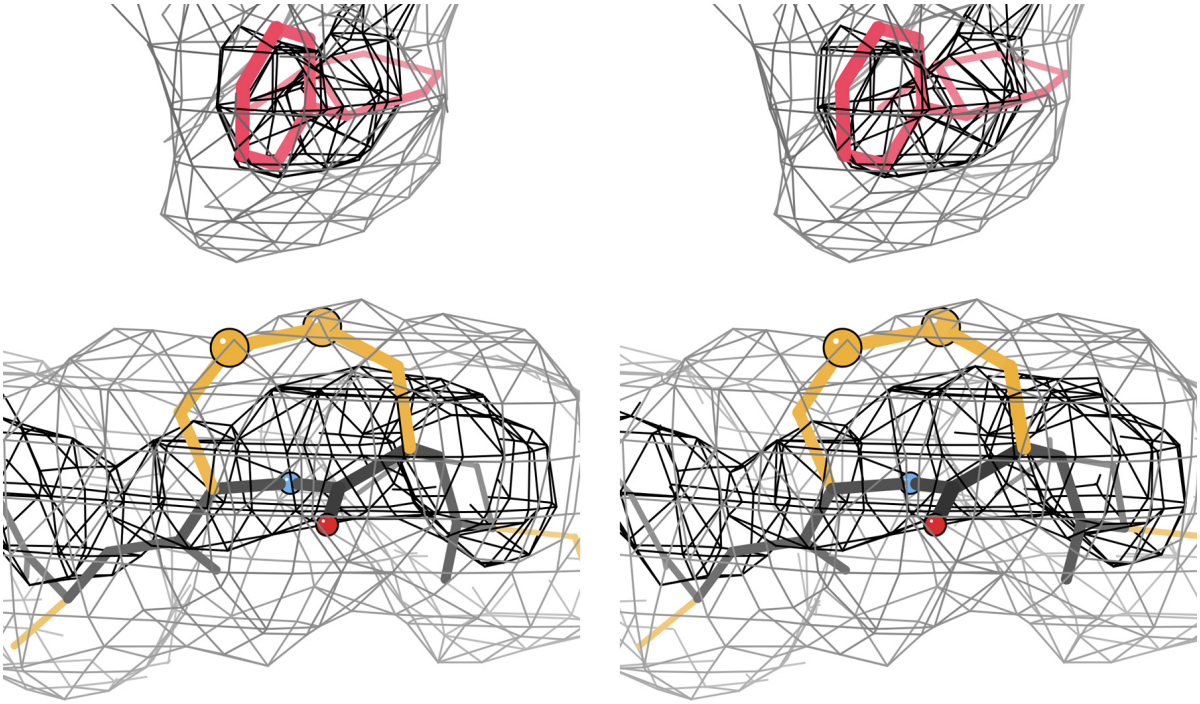


Figure 4: Stereo of the *trans* vicinal disulfide of nAchR at 3.7Å. Cys e 28 from the 6cnj 2α3β nicotinic acetylcholine receptor (Walsh 2018).

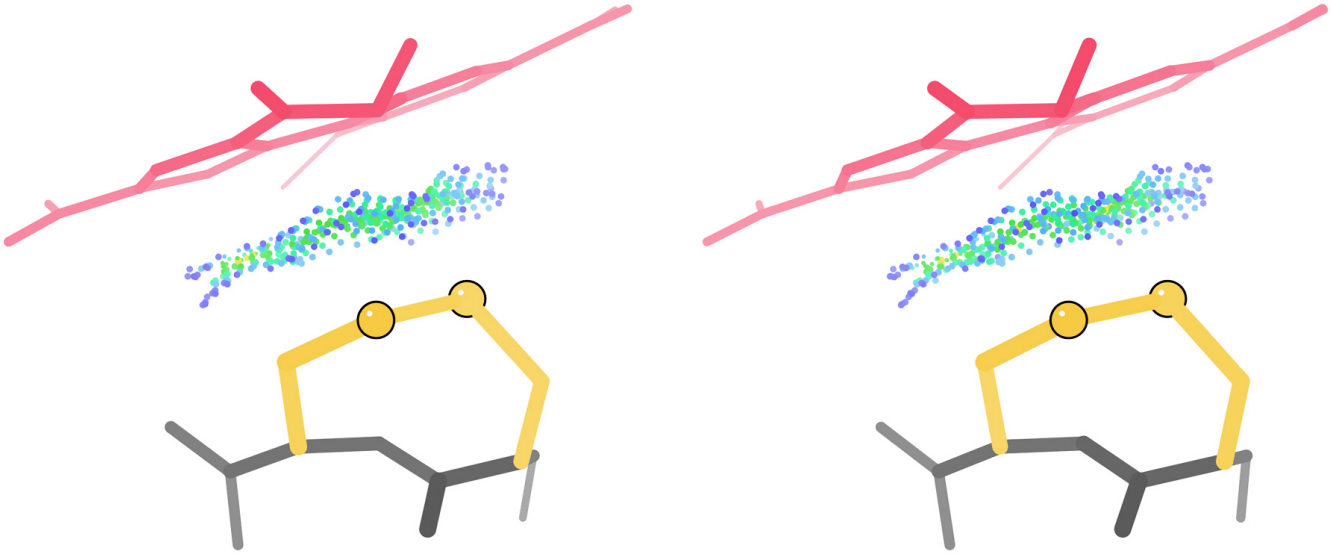


Figure 5: Stereo of a vicinal SS contributing to specific binding of a ring ligand. Cys 104 and PQQ (pyrroloquinoline quinone) from the 2.2Å 2ad6 methanol dehydrogenase (Li 2011). All-atom contacts for favorable van der Waals interaction between SS and PQQ are shown by green and blue dot surfaces.

Other vicinal SS that control large conformational changes occur in von Willebrand blood clotting factor (3gxb) where the SS stays bonded as in nAchR, and in mercuric reductase (1zk7) and the human transglutaminase 2 implicated in celiac disease, where the change involves reduction of the SS bond (2q3z).

Many examples seem to primarily add stability by tight, buried contacts of protein structure around the large lump of the vicinal SS, for example in transferrin-binding

proteins (3hoL), a cytokine receptor (4nn5), and a viral-envelope ribonuclease (4dvk).

Perhaps the most widespread but unexpected function of vicinal disulfides is to provide specific binding of the undecorated face of a sugar or other ring ligand. This occurs in at least six unrelated protein families with distinct folds: nAchR (2qc1), ConA-type lectins (1wd3), small Greek-key antiparallel β (1k12), β-propeller-5 (1gyh, 3d61), β-propeller-8 (2ad6), and TIM barrel (2fhf). The flat, rigidly held C-S-S face of the vicinal SS makes extensive steric contact with the ligand ring, thus selecting for an undecorated position with only H, in concert with H-bonding groups that can select for a position with an OH group. Figure 5 shows the all-atom contact dots of such a case, for Cys-*trans*-Cys104 helping bind the PQQ cofactor in the 2ad6 methanol dehydrogenase at 2.2Å (Li 2011). Functional details about more vicinal SS examples can be found in Richardson 2017.

The bottom line

Vicinal disulfides are extremely rare; so don't model one unless you're sure it's right. But if you do see one, it's very likely to be of functional importance for stability, or for control of conformational change, or for specific binding of the undecorated face of a ring ligand.

References:

Carugo O, Cemazar M, Zahariev S, Hudaky I, Gaspari,Z, Perczel,A, Pongor S (2003) Vicinal disulfide turns, *Protein Engin,* **16**: 637-639

Chandrasekharan R, Balasubramanian R (1969) Stereochemical studies of cyclic peptides. VI. Energy calculations of the cyclic disulphide cysteinyl-cysteine, *Biochim Biophys Acta* **188**: 1-9

Creighton CJ, Reynolds CH, Lee DHS, Leo GC., Reitz AB (2001) Conformational analysis of the eight-membered ring of the oxidized cysteinyl-cysteine unit implicated in nicotinic acetylcholine receptor ligand recognition, *J Am Chem Soc.***123**: 12664-12669

Li J, Gan J-H, Mathews FS, Xia Z-X (2011) The enzymatic reaction-induced configuration of the prosthetic group PQQ of methanol dehydrogenase, *Biochem Biophys Res Commun* **406**: 621-626 [2ad6]

Miyanaga, A., Koseki, T., Matsuzawa, H., Wakagi, T., Shoun, H. & Fushinobu, S. (2004). Crystal structure of a family 54 alpha-L-arabinofuranosidase reveals a novel carbohydrate-binding module that can bind arabinose, *J Biol Chem* **279**: 44907-44914 [1wd3]

Pancera M, Zhou T, Druz A, Georgiev IS, Soto C, Gorman J, Huang J, Acharya P, Chuang G-Y, Ofek G, Stewart-Jones GBE, Stuckey J, Bailer RT, Joyce MG, Louder MK, Tumba N, Yang Y, Zhang B, Cohen MS, Haynes BF, Mascola JR, Morris L, Munro JB, Blanchard SC, Mothes W, Connors M, Kwong PD (2014) Structure and immune recognition of trimeric pre-fusion HIV-1 Env, *Nature* **514**: 455-461 [4tvp]

Richardson JS, Videau LL, Williams CJ, Richardson DC (2017) Broad analysis of vicinal disulfides: Occurrences, conformations with *cis* or with *trans* peptides, and functional roles including sugar binding, *J Molec Biol* **429**: 1321-1335

Walsh RM, Roh SH, Gharpure A, Morales-Perez CL, Teng J, Hibbs RE (2018) Structural principles of distinct assemblies of the human alpha 4 beta 2 nicotinic receptor, *Nature* **557**: 261-265 [6cnj]

FAQ

How do I model a partially broken disulphide?

Use the phil parameter:

disulfide_bond_exclusions_selection_string="chain A and resseq 34 and name SG and altloc A"

It works in `phenix.refine, phenix.real_space_refine, phenix.dymanics, phenix.geometry_minimization`, and more. There are GUI fields of these commands in "All parameters" or "model interpretation parameters" where a search will find the interface required. There is a video tutorial "Changing custom parameters in phenix.refine" @

http://phenix-online.org/documentation/reference/tutorial_channel.html

CaBLAM: A C-Alpha Based Low-resolution Annotation Method for secondary structure and validation

Christopher J. Williams, David C. Richardson, and Jane S. Richardson
Department of Biochemistry, Duke University, Durham, NC 27710

Correspondence email: Christopher.sci.williams@gmail.com or dcrjsr@kinemage.biochem.duke.edu

Introduction

The intent of this piece is to provide documentation of the CaBLAM validation, including its methods, its parameter space and its accessibility within *Phenix*. More detail is provided here than could be included in the Williams 2018 MolProbity paper; still further background and detail can be found in the Williams 2015 PhD thesis.

CaBLAM stands for C-Alpha Based Low-resolution Annotation Method. It is a validation system designed to describe and validate protein backbone using Cα geometry and relative peptide plane orientations. CaBLAM is most valuable as a validation at resolutions and in regions where the backbone Cα trace can be reasonably determined from the electron density, but the positions of backbone carbonyl oxygens cannot.

CaBLAM measures

CaBLAM describes protein backbone using various 2- and 3-dimensional parameter spaces constructed from 4 geometric measures. These measures are two Cα pseudodihedrals, μ_{in} and μ_{out}; the Cα virtual angle; and a dihedral relating adjacent peptide planes, ν.

The two Cα pseudodihedrals, or virtual dihedrals, (figure 1) are used in all of CaBLAM's parameter spaces. μ_{in} is defined using the atom positions $C\alpha_{i-2}$, $C\alpha_{i-1}$, $C\alpha_i$, $C\alpha_{i+1}$. μ_{out} is defined using the atom positions $C\alpha_{i-1}$, $C\alpha_i$, $C\alpha_{i+1}$, $C\alpha_{i+2}$. The combined calculation of μ_{in} and μ_{out} requires five residues in sequence from $C\alpha_{i-2}$ to $C\alpha_{i+2}$, making CaBLAM validation undefined within two residues of chain termini and breaks.

The Cα virtual angle is defined in the conventional manner, using the atom positions $C\alpha_{i-1}$, $C\alpha_i$, $C\alpha_{i+1}$.

The ν dihedral (figure 2) requires the construction of two pseudo-atom points. The point X_{i-1} is constructed on the line from $C\alpha_i$ to $C\alpha_{i-1}$, at the point closest to O_{i-1}. The point X_i is constructed on the line from $C\alpha_i$ to $C\alpha_{i+1}$, at the point closest to O_i. The ν dihedral is then defined using the positions O_{i-1}, X_{i-1}, X_i, O_i.

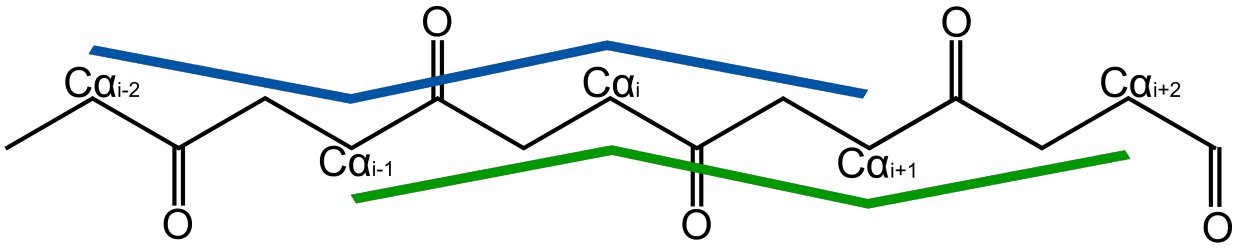


Figure 1: μ_{in} (blue) and μ_{out} (green) pseudodihedrals describe Cα trace of protein backbone.

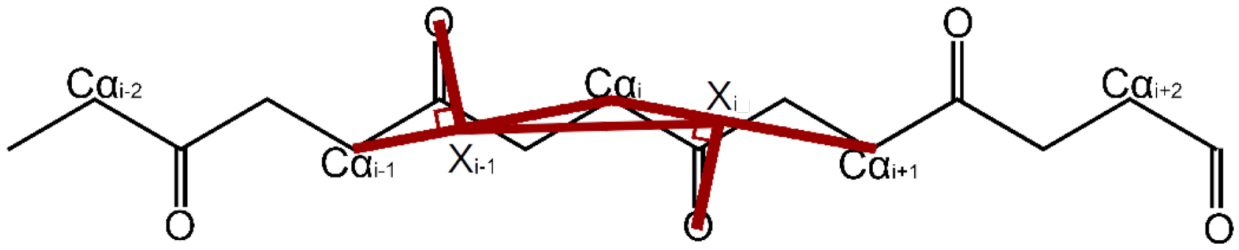


Figure 2: ν pseudodihedral describes torsion relation between adjacent peptide planes across the Cα of the residue of interest.

CaBLAM parameter spaces

Three parameter spaces are constructed from these parameters. In all three parameter spaces, μ_{in} is the x-axis and μ_{out} is the y-axis. The main CaBLAM parameter space uses ν as the z-axis. This space is used to identify CaBLAM outliers. A second, Cα-only space uses the Cα virtual angle as the z-axis. This space is used to identify Cα geometry outliers. The third and final parameter space is two-dimensional, consisting of only μ_{in} and μ_{out}. This two-dimensional space is used to identify secondary structure elements.

Contour levels for these parameter spaces were set using data from the Top8000 quality-filtered database. The $\mu_{in}/\mu_{out}/\nu$ space uses two-tiered cutoffs of 1% for CaBLAM outliers and 5% for CaBLAM disfavored, similar to the outlier and allowed cutoffs in Ramachandran space. The μ_{in}/μ_{out}/Cα-virtual-angle space uses a single cutoff at 0.5% for Cα geometry outliers. The 2D μ_{in}/μ_{out} space uses a cutoff of 0.1% for identifying alpha and 3_{10} helix, and a cutoff of 0.01% for identifying beta strand. The secondary structure behavior is more cleanly defined in CaBLAM space than in Ramachandran space, resulting in steeper edges for the secondary structure distributions and allowing these lower cutoffs without introducing significant false positives.

The geography of the CaBLAM parameter spaces is easiest to understand starting with the 2D μ_{in}/μ_{out} space and its secondary structure contours (figure 3). These representations place 0° at the center of each axis, with -180° at the

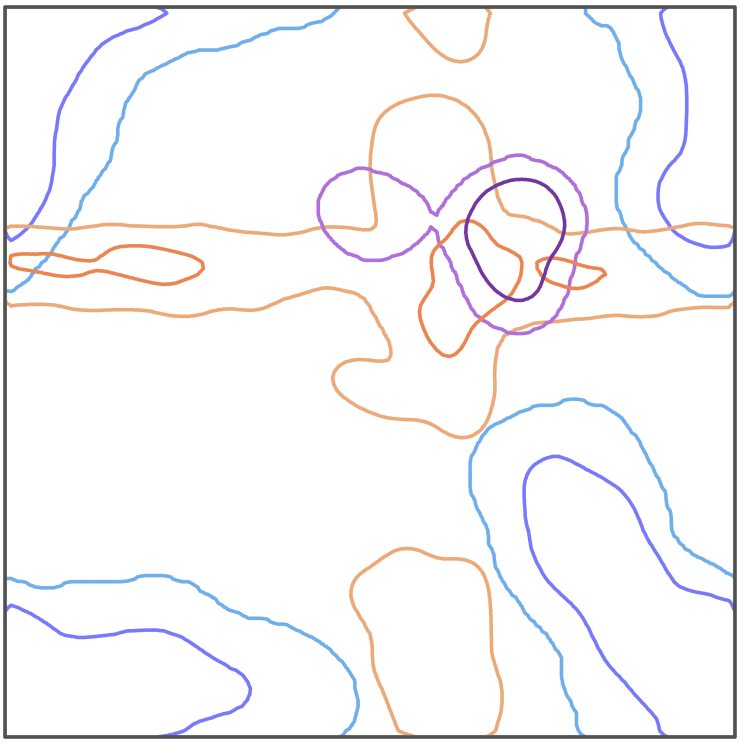


Figure 3: The 2D μ_{in}/μ_{out} space used for secondary structure identification. Cutoffs for alpha helix are shown in orange, 3_{10} helix in purple, and beta strand in blue.

bottom/left and +180° at the top/right. As in the Ramachandran plot, the edges of this space wrap to the other side. The colored outlines in figure 3 show the cutoff contour levels for alpha helix (orange), 3_{10} helix (purple), and beta strand (blue). Linear, elongated structures like beta strand are centered in the corners around ±180°. A completely *cis* Cα conformation – that is, both μ dihedrals at 0° – would fall at the very center of this space. Alpha helix is elongated and twisted slightly from *cis* to permit its repetitive structure, and so the center of the alpha helix distribution is somewhat up and to the right of center. 3_{10} helix requires further elongation of the Cα trace to permit its tighter repeat pattern, and so its center falls further from *cis* than the alpha helix. Right-handed conformations like alpha and 3_{10} helix result in positive μ dihedral values, so these motifs are centered up and right from *cis*. Left-handed alpha would appear in the lower left.

These secondary structure contours also show transitional structures, most clearly evident in the alpha helix contours. The center of the alpha helix distribution has two long arms, a continuous arm across μ_{in} and a discontinuous arm along μ_{out}. These arms represent residues with one helix-like μ and one helix-unlike μ, that is, residues in transition between helix and another structure type such as N- and C-caps. The CaBLAM parameter space's ability to represent transitional structures is both a benefit of the system and a challenge that must be managed when interpreting its results.

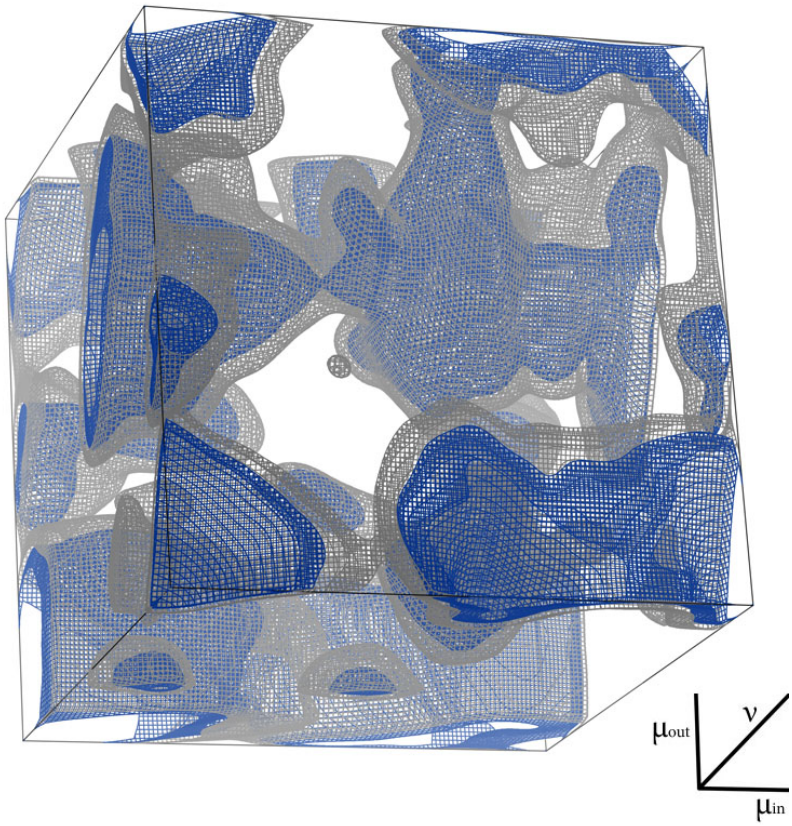


Figure 4: The 3D $\mu_{in}/\mu_{out}/\nu$ space used to identify CaBLAM outliers. The 10% contour is shown in blue and the "disfavored" 5% contour in gray.

The 3D $\mu_{in}/\mu_{out}/\nu$ space (figure 4) is dominated by the common secondary structure elements, which are further differentiated by their distribution in the ν dimension. In beta structure, carbonyl oxygens alternate direction along the strand, near 180° from each other, so beta strand residues are clustered in the corners of this space. In alpha helix, successive carbonyl vectors are only about 60° away from *cis*, so alpha helix residues cluster just above the center of the ν axis. Some new clusters become evident in the full 3D space. The most interesting of

these are the clusters at the very center of each μ/ν face of the parameter space cube, seen most clearly on the μ_{in}/ν face at the bottom of figure 4. These clusters are where beta bulges fall, having both ν and one μ near *cis*. The distance between the beta strand clusters in the corners and the beta bulge clusters on the faces demonstrates the surprising distance that can occur between related structures in the CaBLAM parameter space.

The 3D μ_{in}/μ_{out}/Cα-virtual-angle space (figure 5) is much more compressed, because the Cα virtual angle for good data samples only a range of about 70°-160°. Secondary structures are not distinctly visible in this distribution, although beta strands are extended and thus have larger Cα virtual angles than helices. At the contour level (0.5%) used for identifying outliers, virtually all μ_{in}/μ_{out} combinations are permitted. Therefore, most Cα geometry outliers are assumed to be problems with the Cα virtual angle. Inspection of structures supports this assumption, as most Cα geometry outliers involve obviously too-extended or too-constricted Cα virtual angles.

As in Ramachandran analysis, proline residues have a significantly more restricted distribution than the general case and glycine residues have a significantly more permissive distribution. Proline and glycine therefore each have their own sets of 3D contours for use in validation. Further subcategories of residue types are not currently defined for CaBLAM.

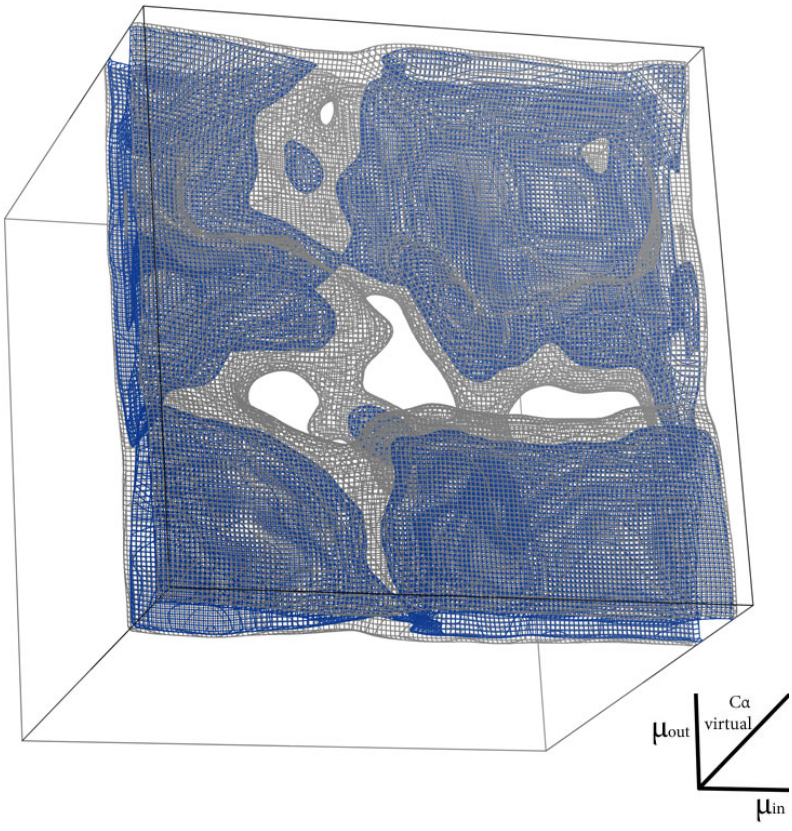


Figure 5: 3D μ_{in}/μ_{out}/Cα virtual angle space used to identify Cα geometry outliers. The 5% contour is shown in blue and the 1% contour in gray.

Interpretation, and assembly of secondary structure

When CaBLAM validation is run, each residue is scored against the 3D CaBLAM contours, the 3D Cα contours, and the 2D contours for alpha helix, 3_{10} helix, and beta strand. Residues that fall below the cutoffs in the 3D CaBLAM or Cα spaces are marked as outliers. Residues that fall above the cutoffs for secondary structure become candidates for assembly into secondary structure elements. A candidate residue is identified as beta strand if that residue and both the preceding and succeeding

residues all pass the beta strand cutoff. Helices can transition between alpha and 3_{10}, so a candidate residue is identified as alpha helix if that residue passes the alpha helix cutoff, and both the preceding and succeeding residues all pass either the alpha or 3_{10} helix cutoff. A candidate residue is identified as 3_{10} helix if it passes the 3_{10} helix cutoff and it scores higher for 3_{10} than for alpha plus at least one of the adjacent residues also passes the 3_{10} helix cutoff. If all of these conditions are met, identification as 3_{10} will override identification as alpha.

Adjacent residues that share the same secondary structure identification are assembled into secondary structures: alpha helix, 3_{10} helix and beta strand. Individual beta strands are not currently assembled into beta sheets because proper registration of strands is challenging in structures where hydrogen bonding is not reliable.

Accessing CaBLAM in *Phenix*

CaBLAM is accessible through the commandline as `phenix.cablam`. The commandline accepts a single PDB or mmCIF file. The default output is a text validation of each residue in the structure, plus a summary of the overall structure statistics. Other outputs can be accessed through the `output=` flag, the most significant of which are `output=kin` for printing CaBLAM's kinemage markup and `output=records` for printing ksdssp-style HELIX and SHEET records.

Internally, CaBLAM is structured similarly to our other validation scripts like ramalyze and rotalyze with a validation class named `cablamalyze` that accepts a model hierarchy object plus some other arguments for controlling amount and destination of output (`sys.stdout`, by default). The `cablamalyze` object contains a list, `cablamalyze.results`, of validations for each residue. In a result object, the CaBLAM geometry parameters can be found in `result.measures`, the contour scores in each parameter space in `result.scores` and the assessment of whether that residue is an outlier and/or secondary structure in `result.feedback`. The `cablamalyze` object contains functions to return various structure-level summary statistics such as available from the parent class `percent_outliers()` function. Another class function, `as_secondary_structure()`, will assemble CaBLAM validations into secondary structure elements and return a *Phenix*-compatible secondary structure annotation object.

Kinemage markup

CaBLAM provides three forms of visual markup for validation kinemages. The first two (figure 6) mark outlier and disfavored residues in CaBLAM space. These markups

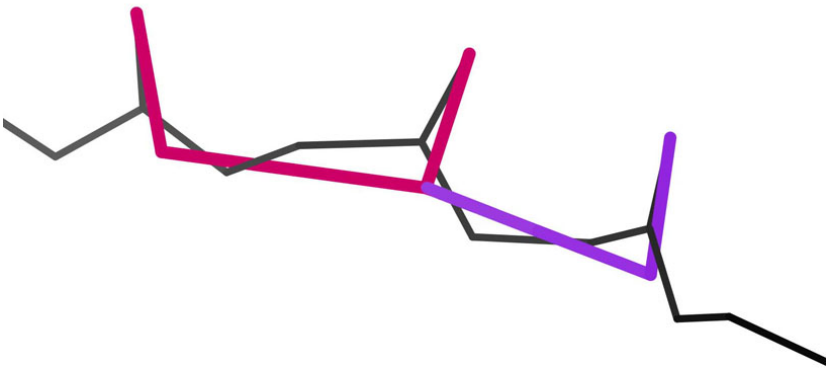


Figure 6: Kinemage markup of an outlier (pink) or disfavored (purple) residue traces the ν dihedral of that residue.

trace the ν dihedral for the outlier or disfavored residue, as that is the geometry most likely to be in error. Disfavored residues (bottom 5% of reference-data protein behavior) are marked in purple and outlier residues (bottom 1% of reference-data protein behavior) in hot pink.

The third markup (figure 7) shows Cα geometry outliers in the μ_{in}/μ_{out}/Cα-virtual-angle space. The Cα geometry markup is red and follows the Cα virtual angle that is both the measure unique to the Cα geometry validation and the parameter most likely to be in error in a Cα geometry outlier.

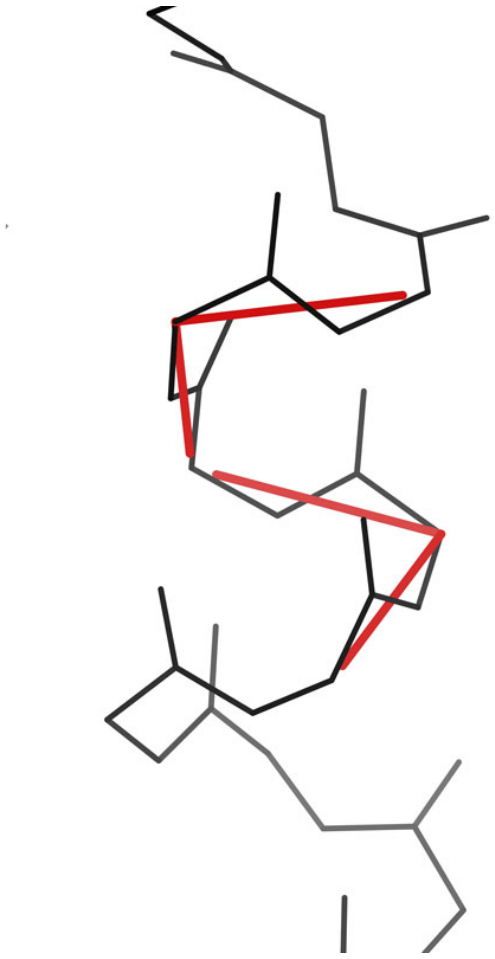


Figure 7: Kinemage markup of Cα geometry outliers follows the Cα virtual angle.

Secondary structure annotations made by CaBLAM are not presented with visual markup as such. However, selecting any vertex of the CaBLAM or Cα geometry markup will display text that includes the secondary structure scores for that residue and KiNG can draw secondary structure ribbons based on HELIX and SHEET records generated by CaBLAM. The secondary structure annotations are designed to be conservative with high confidence and few false positives.

Notes on best usage and interpretation

CaBLAM is bootstrapping up from a minimal form of information – the Cα trace – that is relatively reliable even in poor models. This makes CaBLAM invaluable in validating structures that other methods cannot reliably assess. However, above a certain level of structure quality, more sensitive and precise validations are likely to provide more value and nuance. In MolProbity, we currently approximate this level of structure quality with a resolution cutoff of 2.5Å, and CaBLAM validation is automatically enabled for structures at this resolution or worse.

CaBLAM was designed to address a difficult problem – providing useful validation for structures with minimal reliable information. As a result, some care must be taken in interpreting its feedback. CaBLAM uses two cutoffs to provide some nuance to its validation. The 1% (outlier) cutoff misses some clear modeling problems, while the 5% (disfavored) cutoff includes some motifs that are clearly real-but-rare. No single cutoff adequately separates good structure from bad in CaBLAM space, so context becomes vital.

CaBLAM is overly sensitive in loop regions due to their high variability, so disfavored validations, and sometimes even outliers, in loops should be regarded only as general areas for improvement. However, regular regions, such as those in which CaBLAM has identified secondary structure, disfavored validations do represent significant departures from expected protein behavior and should be taken as serious guides for improvement.

The most generally useful model-rebuilding guidance to take from CaBLAM validation is to:

1) build ideal secondary structure where annotated, and
2) try rotating peptides at one side or the other of a CaBLAM outlier to optimize H-bonding while avoiding steric clashes and other kinds of outliers.

References

Williams CJ, (2015) Using C-alpha geometry to describe protein secondary structure and motifs, Duke University PhD dissertation, 248 pages.

Williams CJ, Hintze BJ, Headd JJ, Moriarty NW, Chen VB, Jain S, Prisant MG, Lewis SM, Videau LL, Keedy DA, Deis LN, Arendall WB III, Verma V, Snoeyink JS, Adams PD, Lovell SC, Richardson JS, Richardson DC (2018) MolProbity: More and better reference data for improved all-atom structure validation, *Protein Science* **27**:293-315.

Tools for interpreting cryo-EM maps using models from the PDB

Tom Terwilliger

Los Alamos National Laboratory, Los Alamos NM 87545
New Mexico Consortium, 100 Entrada Dr, Los Alamos, NM 87544

Phenix now has a set of tools for finding the symmetry in a cryo-EM map, cutting out the unique part of a map and docking a model into a map. These tools now make it easier for you to interpret a cryo-EM map using existing models from the PDB.

Finding symmetry in a map with *phenix.map_symmetry*

The *phenix.map_symmetry* tool is designed to find the reconstruction symmetry used to create a cryo-EM map and to write out a file containing the symmetry operators. The tool has an internal database of common symmetries and it quickly checks to see which ones match the map that you supply. It makes the assumption that the principal axes of symmetry often match the *a,b,c* axes of your map. For helical symmetry, the tool assumes that the helix is along the *z*-axis.

You can run the map symmetry tool with a simple command such as:

```
phenix.map_symmetry emd_8750.map symmetry=D7
```

This will look for D7 symmetry (7-fold symmetry about *z* and 2-fold symmetry about *x* or *y*) and it will report back the symmetry operators that match the map. You can leave off the `symmetry=D7` keyword and the map symmetry tool will look for all types of symmetry.

The symmetry operators that you find with *phenix.map_symmetry* can be useful later if you want to create a complete molecule from one component chain. You might want to do this if you work on a single chain, for example. You can create the entire molecule with:

```
phenix.apply_ncs edited_chain_A.pdb symmetry_from_map.ncs_spec
```

This command will apply each symmetry operator in *symmetry_from_map.ncs_spec* to the chain or chains in edited_chain_A.pdb and create a new model with all these chains. New chain ID will be created for the new chains.

Cutting out the unique part of a map

The *phenix.map_box* tool is a multi-purpose tool that allows you to make a new smaller cryo-EM map by cutting out a part of your cryo-EM map. A new option for the *phenix.map_box* tool is to automatically find and cut out the unique part of your map. All you need to supply is your map, the resolution, the molecular mass of the contents of your map and the symmetry or a symmetry file with your symmetry operators.

The way this works is the *phenix.map_box* tool tries to find a compact part of your map that, including your symmetry operators, represents the whole map. This is done by first finding all the regions in your map that are above a certain contour level, then finding which of these are

duplicated by the map symmetry and choosing a set of regions that is unique and compact. The method is described in Terwilliger et al., 2018.

Figure 1 shows an example of applying the map-box tool in this way. The map is the deposited cryo-EM map of groEL (EMD entry 8750) and the command used is:

```
phenix.map_box emd_8750.map extract_unique=true resolution=4 \
molecular_mass=1000000 symmetry=d7
```

Figure 1 shows the entire map in purple the extracted part of the map in yellow. The extracted part very closely matches chain G of the model for this map (pictured in Fig. 1; PDB entry 5w0s).

Docking a model into a cryo-EM map

You can dock a model into a cryo-EM map using the new *Phenix* tool, *phenix.dock_in_map*. This tool finds the translation and rotation that best matches your model to the map. If you have

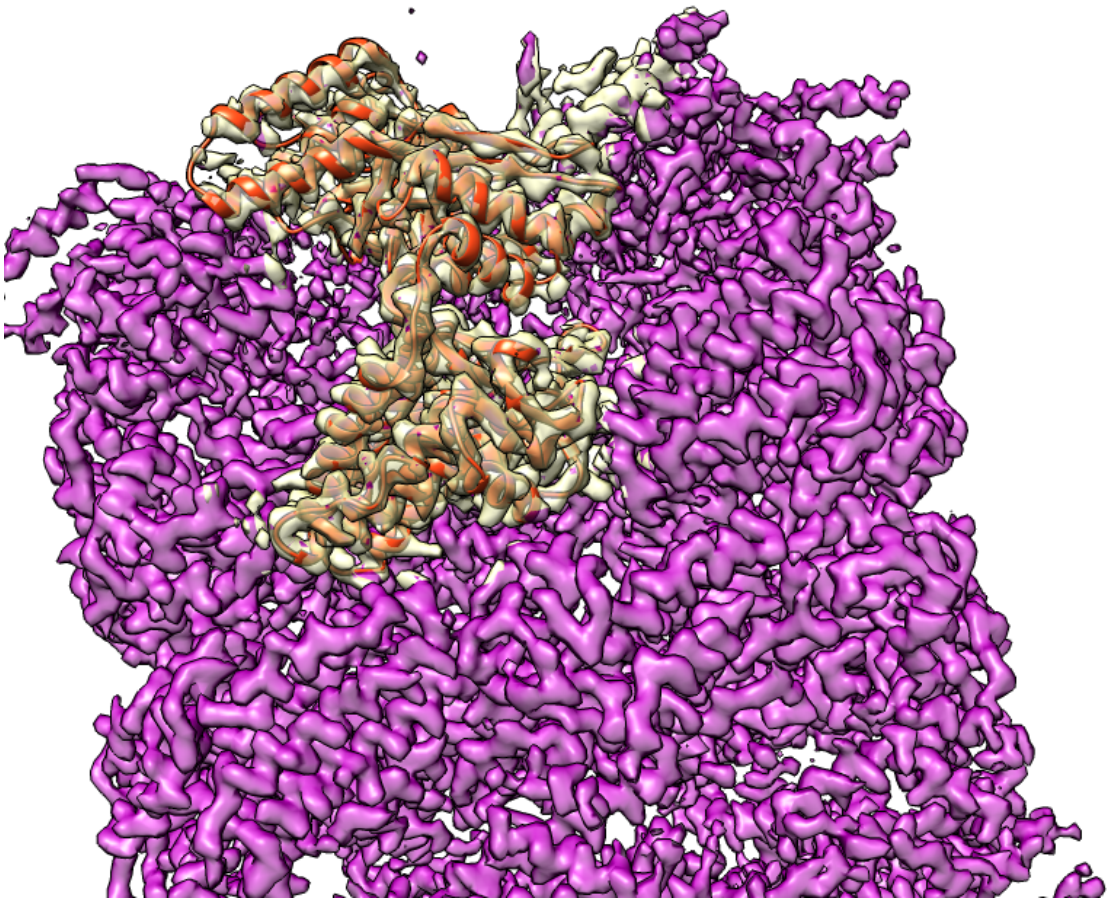


Figure 1

more than one model, or want to fit more than one copy of your model in the map, you can do those things as well.

The *phenix.dock_in_map* tool often can find the location of your model very quickly because it first uses low-resolution representations of your model and the map to find the placement and orientation of the model. Then if the placement is satisfactory, real-space rigid-body refinement is carried out using the full resolution of the map to optimize the placement of the model.

If you are placing more than one model, the density for all previously placed models is first removed from the map, then a search is carried out for the next model to be placed. This can allow you to construct a complex molecule from its parts.

Figure 2 shows how you can dock a model of groEL for chain A from the PDB entry 1ss8 into the deposited full cryo-EM map shown in purple in Fig. 1. The command used is:

```
phenix.dock_in_map 1ss8_A.pdb emd_8750.map resolution=4 nproc=4 \
pdb_out=placed_model_from_emd_8750.pdb
```

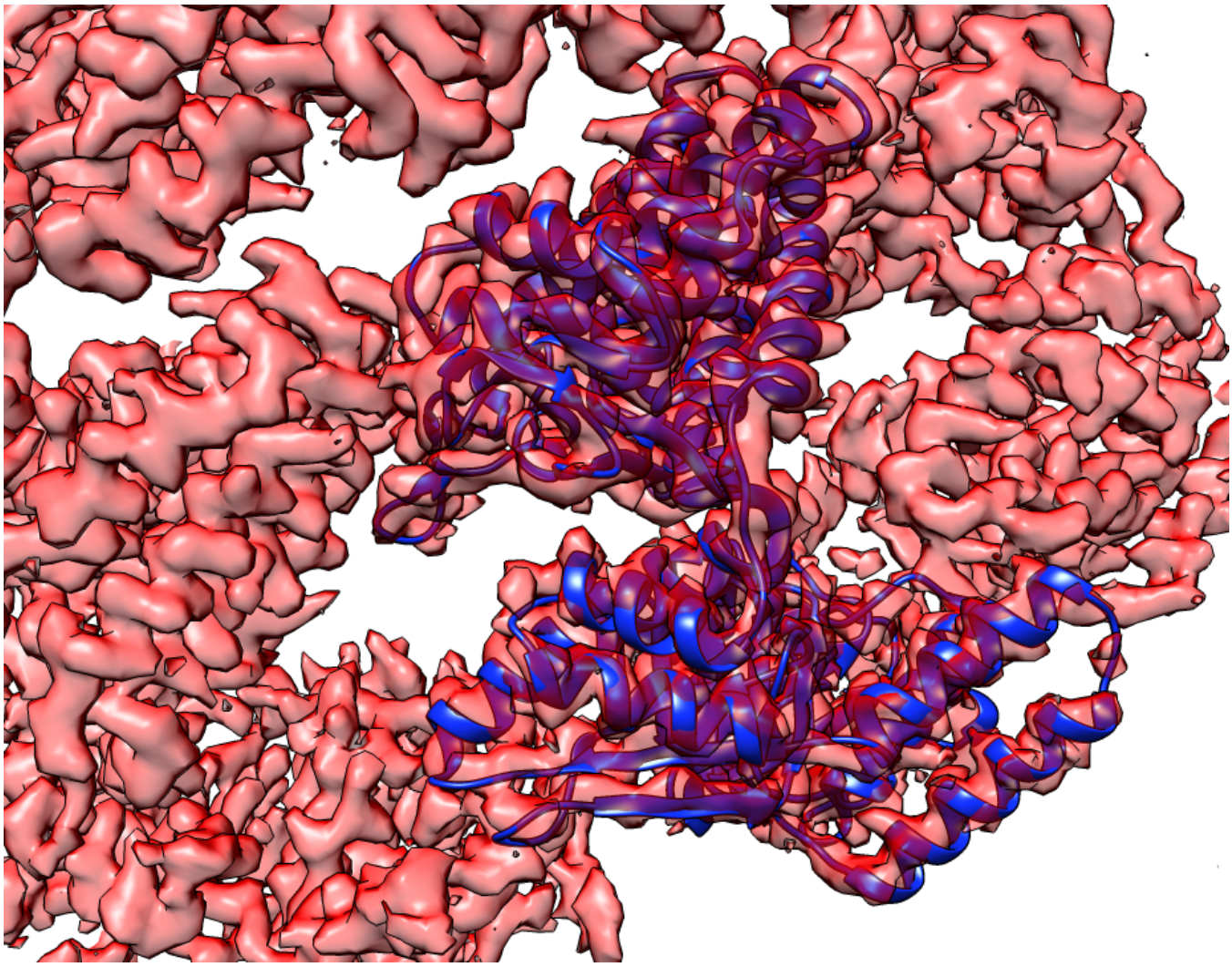


Figure 2

In figure 2 you can see the density for the groEL map in pink and the docked model for 1ss8 chain A in blue.

You can do all these things yourself in a few minutes using the instructions and data in the tutorial called "groel_dock_refine".

Reference:

Terwilliger, T.C., Adams, P.D., Afonine, P.V., Sobolev, O.V.(2018). "Map segmentation, automated model-building and their application to the Cryo-EM Model Challenge" BioRxiv doi: https://doi.org/10.1101/310268

Using the New Program Template

Billy K. Poon

Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Correspondence email: BKPoon@lbl.gov

Introduction

To help unify the command-line and graphical interfaces for CCTBX-based programs (e.g. Phenix), a new approach based on a program template is introduced. This approach provides some consistent functionality for basic tasks, like file and parameter handling, as well as helps ensure that the same exact code is executed regardless of the user interface.

At a high level, this new approach is based on the Model-View-Controller (MVC) design pattern, probably first introduced by Trygve Reenskaug at Xerox PARC in the late 1970's [1, 2]. Generally, the end user interacts with the View, which can be the command-line terminal or a graphical user interface (GUI). The Controller serves as a translation layer that can convert the user interactions into something the Model uses for the actual work and can convert the output from the Model into something displayed by the View that is understandable by the end user. In this new approach, the Model is the core library functionality of CCTBX that developers use for calculations, the View is the command-line or GUI that end users interact with, and the Controller is composed of several new classes that more clearly define the boundary between the user and the underlying scientific code. These classes are the DataManager that keeps track of the mapping between user-provided data files and the resulting CCTBX data structures that developers manipulate, the ProgramTemplate that explicitly defines a series of steps that a program goes through, and the CCTBXParser that provides a consistent command-line interface (resoectively?). Figure 1 summarizes the relationships between the MVC and the new classes.

Generally, the CCTBXParser takes command-line input from the user to construct a `DataManager` object that contains the input data (e.g. files) and the regular PHIL scope

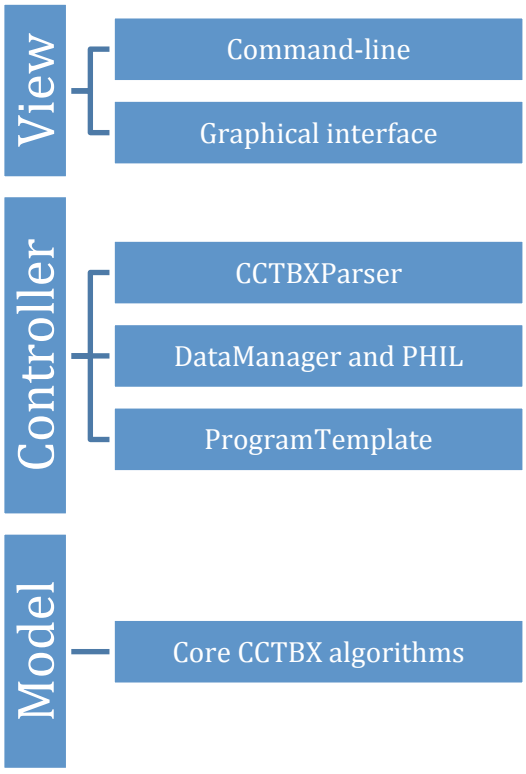


Figure 1: Model-View-Controller design pattern and new classes. User input from the command-line is translated by the CCTBXParser into DataManager and PHIL objects, which are used to create a program object based on the ProgramTemplate. The program then calls core CCTBX functions to do work before reporting output back to the user.

extract object that contains program parameters (e.g. resolution=2.0). These two objects are then used to construct an instance of a program based on the ProgramTemplate since they fully define a job. Once instantiated, standard function names defined in the ProgramTemplate are called to perform the actual work and display output to the terminal. A new graphical interface is under development that utilizes the DataManager and ProgramTemplate classes and will be the graphical equivalent to the CCTBXParser.

New Classes:

To help familiarize developers with this new approach, there is a description of each new class below with some code samples. These code samples are part of a non-trivial program (EMRinger) that was reorganized to adopt this approach. Figures 2 and 3 show the full source code for the program class (mmtbx/programs/emringer.py) and a minimal version of the command-line tool (mmtbx/command_line/emringer.py), respectively.

ProgramTemplate (libtbx/program_template.py):

The ProgramTemplate class should be the parent class for future CCTBX programs. A program is narrowly defined as any tool that is made available to the user. That is, a user provides some form of input, either files and/or parameters, and the program returns some sort of output for the user. To perform this function, the program is broken up into discrete stages, listed below with their associated function names.

1. Initialization (__init__) – This is the normal constructor for a class. The required inputs include a DataManager object and a PHIL extract object. These two objects define the input data and input parameters, respectively. The constructor is predefined in the ProgramTemplate class and generally should not be overwritten by children classes. There are optional parameters for the master PHIL (master_phil) and a logging object (logger), but they are automatically filled in by CCTBXParser (and the future GUI). The constructor basically just sets the arguments to attributes of the instance (self.data_manager, self.master_phil, self.params, self.logger).
2. Custom initialization (custom_init) – This is an optional function that is called after the normal constructor to do any custom initialization. Generally, this is not required because the initialization step should be lightweight and not do any actual work.
3. Input validation (validate) – This step is required and validates the input provided by the user. Since the DataManager object and PHIL object are stored in self.data_manager and self.params, respectively, the developer can check if there is sufficient information to proceed. For example, in Figure 2, the validate step is highlighted pale blue.

   The EMRinger program takes a model and a map and calculates a score for determining the correct sidechain position [3, 4]. To do this, a model and some form of map (real map or map coefficients) are required. The DataManager object is used to check for the existence of necessary data (self.data_manager.has_models, self.data_manager.has_real_maps, and self.data_manager.has_map_coefficients), but also checks that only one kind of map is provided. The DataManager contains some basic functions for these checks and will be described in greater detail later. There are no checks for parameters because the default values are fine, but if some parameter is required, the check should be done here. For example, if a map resolution

```
from __future__ import division, print_function
try:
  from phenix.program_template import ProgramTemplate
except ImportError:
  from libtbx.program_template import ProgramTemplate
import os
import libtbx.phil
from libtbx.utils import Sorry
from libtbx import easy_pickle
import mmtbx.ringer.emringer
# =============================================================================
program_citations = libtbx.phil.parse('''
citation {
  article_id = emringer1
  authors = Barad BA, Echols N, Wang RY, Cheng Y, DiMaio F, Adams PD, Fraser JS
  title = Side-chain-directed model and map validation for 3D Electron Cryomicroscopy.
  journal = Nature Methods
  volume = 10
  pages = 943-46
  year = 2015
  doi_id = "10.1038/nmeth.3541"
  pmid = 26280328
  external = True
}
citation {
  article_id = emringer2
  authors = Lang PT, Ng HL, Fraser JS, Corn JE, Echols N, Sales M, Holton JM, Alber T
  title = Automated electron-density sampling reveals widespread conformational polymorphism in ...
…
}
''')
# =============================================================================
master_phil_str = '''
include scope libtbx.phil.interface.tracking_params
include scope mmtbx.ringer.emringer.master_params
map_label = 2FOFCWT,PH2FOFCWT
  .type = str
  .input_size = 200
  .short_caption = 2Fo-FC map labels
  .help = Labels for 2Fo-Fc map coefficients
show_gui = False
  .type = bool
output_base = None
  .type = str
output_dir = None
  .type = path
  .short_caption = Output directory
quiet = False
  .type = bool
  .short_caption = no graphs
  .help = Don't output files or graphs
'''
# =============================================================================

class Program(ProgramTemplate):
  description = '''
Program for calculating the EMRinger score.

Minimum required inputs:
  Model file
```

Figure 2: Program class for EMRinger (mmtbx/programs/emringer.py)

```
  Map file (or file with map coefficients)

How to run:
  phenix.emringer model.pdb map.ccp4
'''
    datatypes = ['model', 'real_map', 'phil', 'map_coefficients']
    citations = program_citations
    master_phil_str = master_phil_str
    # ---------------------------------------------------------------------------
    def validate(self):
      print('Validating inputs', file=self.logger)
      self.data_manager.has_models(raise_sorry=True)
      if not (self.data_manager.has_real_maps() or
          self.data_manager.has_map_coefficients()):
        raise Sorry("Supply a map file or a file with map coefficients.")
      elif (self.data_manager.has_real_maps() and
          self.data_manager.has_map_coefficients()):
        raise Sorry("Supply either a map file or a file with map coefficients.")
    # ---------------------------------------------------------------------------
    def run(self):
      map_inp = None
      miller_array = None

      print('Using model: %s' % self.data_manager.get_default_model_name(),
        file=self.logger)
      model = self.data_manager.get_model()

      if self.data_manager.has_map_coefficients():
        miller_arrays = self.data_manager.get_miller_arrays()
        miller_array = self.find_label(miller_arrays = miller_arrays)
        print('Using miller array: %s' % miller_array.info().label_string(),
          file=self.logger)
      elif self.data_manager.has_real_maps():
        print('Using map: %s' % self.data_manager.get_default_real_map_name(),
          file=self.logger)
        map_inp = self.data_manager.get_real_map()
        print("CCP4 map statistics:", file=self.logger)
        map_inp.show_summary(out=self.logger, prefix="  ")

      if (self.params.output_base is None) :
        pdb_base = os.path.basename(self.data_manager.get_default_model_name())
        self.params.output_base = os.path.splitext(pdb_base)[0] + "_emringer"

      if not self.params.quiet:
        plots_dir = self.params.output_base + "_plots"
        if (not os.path.isdir(plots_dir)) :
          os.makedirs(plots_dir)

      task_obj = mmtbx.ringer.emringer.emringer(
        model        = model,
        miller_array = miller_array,
        map_inp      = map_inp,
        params       = self.params,
        out          = self.logger)
      task_obj.validate()
      task_obj.run()
      self.results = task_obj.get_results()

      ringer_result = self.results.ringer_result

      if not self.params.quiet:
        # save as pickle
```

Figure 2: Program class for EMRinger (mmtbx/programs/emringer.py) (continued)

```
      easy_pickle.dump("%s.pkl" % self.params.output_base, ringer_result)
      print ('Wrote %s.pkl' % self.params.output_base, file=self.logger)
      # save as CSV
      csv = "\n".join([ r.format_csv() for r in ringer_result])
      open("%s.csv" % self.params.output_base, "w").write(csv)
      print ('Wrote %s.csv' % self.params.output_base, file=self.logger)

    scoring_result = self.results.scoring_result
    scoring_result.show_summary(out = self.logger)

    #rolling_result = self.results.rolling_result

  # It would be good to have central code for this
  # ---------------------------------------------------------------------------
  def find_label(self, miller_arrays):
    best_guess = None
    best_labels = []
    all_labels = []
    miller_array = None
    for array in miller_arrays:
      label = array.info().label_string().replace(" ", "")
      if (self.params.map_label is not None):
        if (label == self.params.map_label.replace(" ", "")):
          miller_array = array
          return miller_array
      elif (self.params.map_label is None):
        if (array.is_complex_array()):
          all_labels.append(label)
          if (label.startswith("2FOFCWT") or label.startswith("2mFoDFc") or
              label.startswith("FWT")) :
            best_guess = array
            best_labels.append(label)
    if (miller_array is None):
      if (len(all_labels) == 0) :
        raise Sorry("No valid (pre-weighted) map coefficients found in file.")
      elif (len(best_labels) == 0) :
        raise Sorry("Couldn't automatically determine appropriate map labels. "+
          "Choices:\n  %s" % "  \n".join(all_labels))
      elif (len(best_labels) > 1) :
        raise Sorry("Multiple appropriate map coefficients found in file. "+
          "Choices:\n  %s" % "\n  ".join(best_labels))
      elif (len(best_labels) == 1):
        miller_array = best_guess
        print("      Guessing   %s   for   input   map   coefficients"%   best_labels[0],
file=self.logger)
        return miller_array

  # ---------------------------------------------------------------------------
  def get_results(self):
    return self.results
```

Figure 2: Program class for EMRinger (mmtbx/programs/emringer.py) (continued)

```
# LIBTBX_SET_DISPATCHER_NAME phenix.emringer
from __future__ import division, print_function

from iotbx.cli_parser import run_program
from mmtbx.programs import emringer

if __name__ == '__main__':
  run_program(program_class=emringer.Program)
```

Figure 3: Command-line tool for EMRinger (cut from mmtbx/command_line/emringer.py)

is required, the code could look like

```
if self.params.resolution is None:
  raise Sorry("Supply a resolution for the map")
```

The goal of this step is to do a quick check of the user-supplied inputs to determine if it is possible to do the desired work.

4. Run calculation (run) – This step is also required and does the actual calculation. This is program-specific, so it can be as simple or complex as necessary to do the work. For EMRinger, another class is constructed to do the calculation. This brings up the important distinction that the program is separate from core algorithmic functionality. The program, narrowly defined again as a tool exposed to the user, should call core classes/functions to do calculations. The core classes/functions should not call programs. Additionally, log output can be sent to self.logger, which is normally a libtbx.utils.multi_out object.
5. Clean up (clean_up) – This is an optional function for cleaning up any temporary files that may have been created.
6. Get results (`get_results`) – By default, None is returned, but this function can be used to return anything of interest. This function will be important for the new graphical interface because this function will be used for determining the output to be displayed.

The ProgramTemplate defines a standard sequence of steps that programs should use to do something for the user. The steps should be generic enough for any task, but by explicitly defining these steps, we can standardize the behavior of user interfaces.

Furthermore, the class for a program should encapsulate all relevant information about itself. To that end, the `ProgramTemplate` defines several class variables for storing that information. Some basic ones are listed below.

1. description – This is a text description about the program. This will be shown as help text to the user, so some basic information about required files is useful.
2. datatypes – This is a list of data types recognized by the DataManager. For EMRinger, the line from figure 2 is highlighted in pale orange.

   This list is used to construct a DataManager object for EMRinger that will only recognize model files (PDB or CIF), real-space maps, PHIL files, and map coefficients (MTZ or CIF). If some other kind of file is supplied, the DataManager will not be able to process it. The CCTBXParser takes this information and displays output about which files are recognized and which files are not used. The goal is to inform the user of any extra files that have no effect in the program.
3. master_phil_str – This is the regular text that defines the PHIL scope for a program. The include keyword is processed, so the program's PHIL scope can be constructed as a collection of other PHIL scopes.
4. citations – This is a PHIL scope object that contains a list of citations following the format defined in libtbx/citations.py. This is useful for citations that do not exist in the central citation database (libtbx/citations.params). For EMRinger, the citations are created by parsing a text string, as shown in figure 2 slightly truncated from the actual file and highlight in pale green.
5. known_article_ids – This is a list of known citations. The known article ids are stored in libtbx/citations.params. This class variable, along with the citations class variable, are used to construct a list of citations for the program.
6. epilog – This is a text string that is shown at the end of help screen on the command-line. For CCTBX, it is defined as shown on the next page.

```
epilog = '''
For additional help, you can contact the developers at cctbx@cci.lbl.gov

'''
```

Since the ProgramTemplate is just a class, other CCTBX-based projects can subclass this class and redefine any defaults. For example, Phenix redefines the epilog and changes the email address to help@phenix-online.org.

When combined with the other new classes, the ProgramTemplate organizes programs into a consistent format with information in predefined categories. This helps ensure that users get a consistent interface. Also, the standard location for programs is <project>/programs. So for the mmtbx subproject, the programs exist in mmtbx/programs. This is separate from the <project>/command_line directory, which will only contain the command-line interface for these programs.

DataManager (iotbx/data_manager/):

The DataManager maps files provided by the user to CCTBX data structures used by developers. The class handles file reading so developers do not have to worry about reading files and trapping IOError for errors in accessing files. Files are categorized by data types, which are basically the type of data in the file (e.g. models, sequences, reciprocal-space data, real space maps, etc.). Currently, the data types recognized by DataManager are listed below.

1. model – model files (PDB or CIF)
2. phil – PHIL files (text)
3. sequence – sequence files (most formats)
4. restraint – restraint files (CIF)
5. ncs_spec – NCS files (text)
6. real_map – real-space maps (CCP4 format)
7. miller_array – general reciprocal-space data (most formats, e.g. MTZ, CIF, …)
8. map_coefficients – subclass of miller_array that has known labels for map coefficients

There are other data types in development, specifically, more subclasses for miller_array to handle intensities, amplitudes and other reciprocal space data.

This class also introduces the idea of a default file, or the first file encountered of each data type. For programs that only need one file of a specific type, developers do not need to specify a PHIL parameter for that file. For more complicated situations where files and parameters need to be mapped to one another, using PHIL parameters to define that mapping is still required. In the EMRinger program, only a model and a map are required, so instead of having to define PHIL parameters for each file, the first model is recognized as the default model and the first map is recognized as the default map. Since the validate step (shown in the above section) checks that only either real-space maps or map coefficients are provided, there is no case where there is a default real-space map and a default map coefficients file. In the event that multiple files of the same type are provided, the EMRinger program displays which files are used in its analysis. Furthermore, the description in the EMRinger program informs the user that one model and one map (real-space or map coefficients) are the inputs.

The DataManager also defines a set of functions for each data type to access the data structures. Some basic functions for the model data type are listed below, but the “model”

part can be replaced with any other data type (e.g. get_real_map instead of get_model).

1. add_model(filename, data) – creates an entry for <filename> that is associated to <data>
2. set_default_model(filename) – sets <filename> as the default for that data type
3. get_model(filename) – returns the <data> associated with <filename>. If <filename> is not provided, the default is returned.
4. get_model_names – returns a list of known filenames for that data type
5. get_default_model_name – returns the default <filename> for that data type
6. has_models(expected_n=1, exact_count=False, raise_sorry=False) – returns True or False. The expected_n parameter can be changed to the minimum number of items expected. If exact_count is set to True, the number of items has to equal expected_n for True to be returned. If exact_count is set to True, an error (libtbx.utils.Sorry) is raised instead of returning False. This is useful in the validate step of the ProgramTemplate for checking if the expected data exist.
7. process_model_file(filename) – tries to read <filename> and store the associated <data> from the file

Some data types may also have other functions specific for that data type. To see the full list of available functions for each data type, the source code can be browsed in the iotbx/data_manager directory. There is a file for each data type where the name is the data type name (e.g. model.py for the "model" data type).

For general developers, the DataManager will already exist and populated with data, so the main functions for interacting with the DataManager are get_model, get_model_names, get_default_model_name, and has_models and their equivalents for other data types. These functions provide the basic means for getting the data structures and the filenames used for creating those data structures.

CCTBXParser (iotbx/cli_parser.py):

The CCTBXParser is the standard interface for parsing command-line input into DataManager and PHIL objects for creating a program (subclass of ProgramTemplate) object. This parser is a subclass of the standard Python class, argparse.ArgumentParser, that is used for parsing command-line arguments. There is another standard Python parser, optparse.OptionParser, but that module is deprecated.

The only required argument for CCTBXParser is the program class and the parser will pull the relevant information defined in that class to build a standard command-line interface. For example, when a user runs phenix.emringer on the command-line, the default output is shown in schema 1.

The description class variable in figure 2 is shown by the parser as help text and the epilog class variable from the Phenix subclass of the ProgramTemplate is shown at the end. If Phenix were not available, the standard ProgramTemplate in libtbx will be used and that epilog will be displayed instead.

The CCTBXParser class also defines a set of default command-line flags. They are explained in the default output, but generally, the flags are related to showing and saving the parameters in the program, overwriting existing files (e.g. PHIL parameter files) and showing citations. Any program-specific setting should not be a command-line flag; they should be PHIL parameters. This ensures that settings can be encapsulated in files,

```
[bkpoon@eeyore:~] phenix.emringer
usage: phenix.emringer [-h] [--show-defaults [{0,1,2,3}]]
                       [--attributes-level [{0,1,2,3}]] [--write-data]
                       [--write-modified] [--write-all] [--overwrite]
                       [--citations [{default,cell,iucr}]]
                       [files [files ...]] [phil [phil ...]]

--------------------------------------------------------------------------------
Program for calculating the EMRinger score.

Minimum required inputs:
  Model file
  Map file (or file with map coefficients)

How to run:
  phenix.emringer model.pdb map.ccp4
--------------------------------------------------------------------------------

positional arguments:
  files                 Input file(s) (e.g. model.cif)
  phil                  Parameter(s) (e.g. d_min=2.0)

optional arguments:
  -h, --help            show this help message and exit
  --show-defaults [{0,1,2,3}], --show_defaults [{0,1,2,3}]
                        show default parameters with expert level (default=0)
  --attributes-level [{0,1,2,3}], --attributes_level [{0,1,2,3}]
                        show parameters with attributes (default=0)
  --write-data, --write_data
                        write DataManager PHIL parameters to file
                        (emringer_data.eff)
  --write-modified, --write_modified
                        write modifed PHIL parameters to file
                        (emringer_modified.eff)
  --write-all, --write_all
                        write all (modified + default + data) PHIL parameters
                        to file (emringer_all.eff)
  --overwrite           overwrite files, this overrides the output.overwrite
                        PHIL parameter
  --citations [{default,cell,iucr}]
                        show citation(s) for program in different formats

--------------------------------------------------------------------------------
For additional help, you can contact the developers at help@phenix-online.org
[bkpoon@eeyore:~]
```

Schema 1: Default output of EMRinger.

which will help with reproducibility of results. By making these options standard for all programs based on the ProgramTemplate, users will more easily find the parameters available to them and be able to save them. In fact, the --write-all flag will save all the PHIL parameters and when given to the program, will reproduce a previous run.

Also by default, the parser will look for files and PHIL parameters (positional arguments) since these are common inputs to CCTBX-based programs. Furthermore, the parser will recognize any PHIL files and apply the settings stored in these files. And if there are any filenames (path PHIL type) in these PHIL files, the files will automatically be added to the DataManager. However, this is not recursive; that is, if the PHIL file provided on the command-line contains a path parameter with a PHIL file, this PHIL file will be added to the DataManager, but any parameters in this

PHIL file will not be processed by the parser.

It is important to note that there is an order of precedence in how settings are applied. Basically, command-line PHIL arguments are applied in the order they are parsed, from left to right, and they override any settings stored in PHIL files, processed in order from left to right. For example, if there were a program called cctbx.test_program, and it was called in the following way,

```
cctbx.test_program settings1.eff settings2.eff phil_param_1=10
    phil_param_2="what" phil_param_1=5
```

the command-line arguments for phil_param_1 and phil_param_2 will override any values in settings1.eff or settings2.eff. Values in settings2.eff will override values in settings1.eff. Finally, because phil_param_1 is specified twice, that parameter will be set to 5, which is the last specification. To avoid confusion, the parser shows the processing in steps and also has a summary of the modified PHIL parameters after all processing is complete.

Summary:

After CCTBXParser completes parsing any files and PHIL parameters from the command-line, DataManager and PHIL objects are created, which can be used to construct the program object. Then, the steps outlined in the ProgramTemplate section can be called in sequence to perform the work. Because these steps will basically be the same for all programs based on the ProgramTemplate, these steps are consolidated into the run_program function in iotbx/cli_parser.py. As shown in figure 3, the file in the command_line directory can be about 5 lines that call run_program with the appropriate program class from the programs directory. This means that the general developer can just focus on writing the program class as a child class of the ProgramTemplate class and use run_program to get a working command-line tool with all the features outlined above.

Future Work:

While the command-line version is working right now, there are other features in development to further improve consistency and simplify development of programs. The main one is a GUI equivalent to CCTBXParser. Because the constructor for programs only requires a DataManager and a PHIL object, there can be a GUI widget for managing files and constructing the DataManager object, and a GUI for changing PHIL parameters and constructing the PHIL object. The GUI can then construct the program object and go through the same steps of the ProgramTemplate. Because the process of running programs is standardized, all programs can have a basic GUI, similar to how CCTXParser provides a common interface for the command-line. Of course, the option for a more customized GUI, especially for presenting output is still available. More importantly, this is how the same code path will be executed regardless of how the user runs a program, command-line or GUI, which ensures that the same result is the same for both interfaces.

Similar to how there is now a default for simple input files for each data type, another planned feature is to provide default names for output files that are based on the program name. This avoids another common PHIL

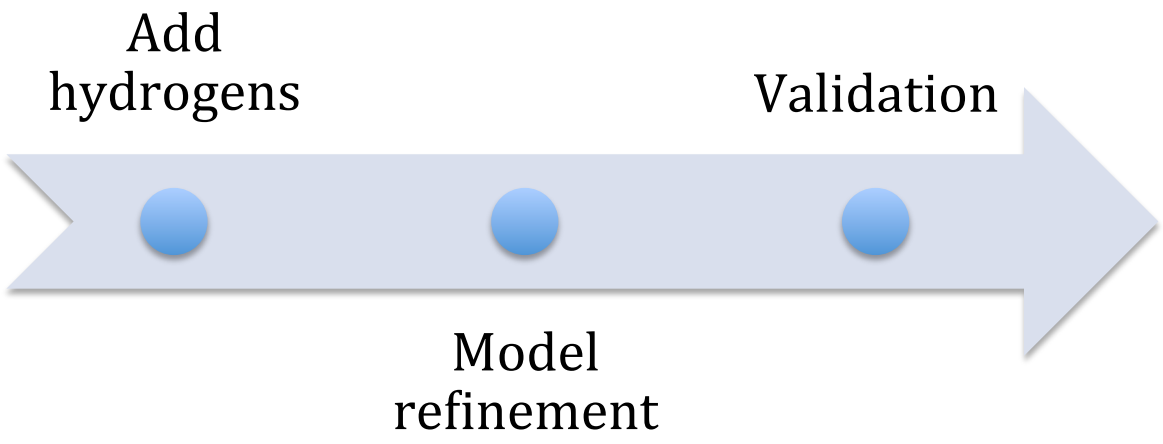


Figure 4: Pipelining example where the blue dot represents the DataManager as it passes through each program. At each step, the default model is updated so that the next step uses the modified model as its input. The "Model refinement" step uses the output of "Add hydrogens," and "Validation" uses the output from "Model refinement." In this case, the data used for refinement and validation is unmodified, so it can just pass through each step normally.

parameter that specifies an output prefix for naming files. Moreover, this should reduce the complexity of developing simple programs that do minor modifications to a file (e.g. converting a model file between PDB and CIF) and just needs to output the result. The DataManager will be responsible for writing output files and this feature will be rolled into the DataManager class. By consolidating the reading and writing of files into the DataManager, the boundary between user files and CCTBX data structures is more clearly defined, which will help with making interfaces more consistent, especially error handling with file input/output.

Longer term, because the DataManager stores information about filenames and data structures for both inputs and outputs, it can be a persistent data repository between programs. That is, developers will be able to link different programs together without having to write scripts or dump data to files between steps because the DataManager has all the relevant information. For example, figure 4 shows a potential pipeline where the first program adds hydrogens to a model file, then the model refinement program is run, and finally validation is performed by the last program on the refined structure. At each step, the default model is updated so that the next step uses the modified model as their default. The "Model refinement" program uses the output from "Add hydrogens," and "Validation" uses the output from "Model refinement." The data used for refinement and validation is unmodified, so it can just pass between steps normally. The data can even be added at the very beginning. The "Add hydrogens" program will just ignore it because the datatypes class variable for that program will not have an appropriate data type and also because the code in the program will not try to do anything with data. Updating the DataManager will probably require the

introduction of a new function to the ProgramTemplate class, most likely called update_data_manager. But because of the nature of classes and inheritance, we can add this function to all programs relatively easily and then customize the implementation for individual programs that need it.

While this is a hypothetical example for pipelining programs, this is currently done in the Phenix GUI for phenix.refine, but in a more *ad hoc* manner and is only available in the GUI. By adopting this new approach, we can standardize the way these pipelines are built and make them available to both the GUI and the command-line. This further improves overall consistency and reduces maintenance overhead by ensuring that the same code path is executed consistently regardless of the interface.

Acknowledgements:

The author wishes to thank members of the Phenix collaboration for helpful discussions about their desired features and requirements for a general program template.

References:

1. Reenskaug, T. "Dynabook System Requirements." 1979 (http://folk.uio.no/trygver/1979/sysreq/SysReq.pdf)
2. Reenskaug, T, Wold, P, and Lehne, OA. "Working with objects - The OOram Software Engineering Method." Manning 1996, ISBN 978-1-884777-10-3, pp. I-XXI, 1-366 (http://heim.ifi.uio.no/trygver/1996/book/WorkingWithObjects.pdf)
3. Barad BA, Echols N, Wang RY, Cheng Y, DiMaio F, Adams PD, Fraser JS. (2015) "Side-chain-directed model and map validation for 3D Electron." Cryomicroscopy. Nature Methods 10:943-46.
4. Lang PT, Ng HL, Fraser JS, Corn JE, Echols N, Sales M, Holton JM, Alber T. (2010) "Automated electron-density sampling reveals widespread conformational polymorphism in proteins." Protein Sci. 7:1420-31.