[*This is a protocol for publication in CPPS. There are 3 figures and 0 tables included. A required PMC file is also included.]

De novo protein design using the blueprint builder in Rosetta

Linna An[1, 2, 3] and Gyu Rie Lee[1, 2, 3]

[1]Institute for Protein Design, University of Washington, Seattle, WA 98195

[2]Corresponding author: linnaan@uw.edu, gyurie@uw.edu

[3]These authors contributed equally to this manuscript.

ABSTRACT:

While native proteins cover diverse structural spaces and accomplish various biological events, not many of them can directly serve human needs. One reason is because the native proteins usually contain idiosyncrasies evolved for their native functions but disfavoring engineering requirements. To overcome this issue, one strategy is to create de novo proteins which are designed to possess improved stability, high environmental tolerances, and enhanced engineering potentials. Compared to other protein engineering strategies, in silico design of de novo proteins significantly expanded the protein structural and sequence spaces, reduced wet lab workload, and incorporated engineered features in a guided and efficient manner. In the Baker laboratory, we have been applying a design pipeline that use the blueprint builder to design different folds of de novo proteins and successfully obtained libraries of de novo proteins with improved stability and engineering potentials. In this protocol, we will use the design of de novo β-barrel proteins as an example to describe the principles and basic procedures of the blueprint builder-based design pipeline.

Basic protocol 1: The construction of blueprints

Basic protocol 2: De novo protein design pipeline using the blueprint builder

Keywords: de novo protein; protein design; blueprint; BluePrintBDR; Rosetta

INTRODUCTION

Designing de novo proteins using Rosetta has successfully provided many robust proteins with diverse topologies for various protein engineering purposes, including small molecule binding (Dou et al., 2018; Tinberg et al., 2013), therapeutic developments (Cao et al., 2020; Fleishman, Whitehead, et al., 2011; Silva et al., 2019), orthogonal biological signaling system construction (Chen et al., 2019; Langan et al., 2019; Quijano-Rubio et al., 2020), and material formation (Hsia et al., 2020; Pyles et al., 2019; Ueda et al., 2020). A number of de novo proteins designed for

above-mentioned applications have significantly different secondary structure features, and were generated using a similar pipeline, which all involved the key step using the structure modifier in Rosetta, the blueprint builder (Huang et al., 2011; Marcos et al., 2018). The name of the modifier in Rosetta is 'BluePrintBDR', and it is usually referred to as the 'blueprint builder'. The protein folds (i.e. the arrangement of a suite of secondary structure elements (SSE, (Schaeffer et al., 2011)) that have been generated include the ferredoxin-like folds (Koga et al., 2012), the Rossmann 2x2 folds (Lin et al., 2015), the TIM barrel folds (Huang et al., 2016), the Nuclear transport factor 2-like protein folds (NTF2-like, (Basanta et al., 2020; Marcos et al., 2018)), the β-barrel folds (Dou et al., 2018), and multiple miniprotein folds (Chevalier et al., 2017). The β-barrel is a family of barrel-like protein structures which are composed of a suite of β-sheets, among which the first strand and the last strand of the β-sheet are connected via backbone hydrogen bonds to form a closed barrel shape. In this protocol, we will adapt the functions of a set of published scripts that were used for generating β-barrel folds as an example (Dou et al., 2018) to explain the idea and the technical details of the blueprint builder-based de novo protein designing pipeline.

As our focus is to introduce the use of the blueprint builder-based pipeline for structure generation, this design pipeline will be referred to as the blueprint pipeline in the coming report. In general, the blueprint pipeline includes three steps, blueprint (and/or a constraint file) construction, protein backbone building following the guidance and restrictions in the blueprint (and/or constraint) file, and protein sequence design on the generated backbones. The blueprints can either be generated based on careful designing of a protein fold from scratch (see Basic Protocol 1), or extracting the structure information from an existing protein of interest (see Alternative Protocol 1, (Huang et al., 2011)). A computational filter step is usually added following the second and the third step, respectively, to supervise the quality of the generated protein backbones and the design of the protein sequences.

By following this protocol, the readers will be able to run the demonstrative scripts on their own, obtain a general understanding of the blueprint pipeline, and have access to all resources needed to start using the required Rosetta functions, including the BluePrintBDR. The authors assume the readers have a little basic knowledge of programming languages and interpreters including python, bash, and xml, or can follow the provided resources to find the needed information. The authors also assume the readers have or can follow the provided resources to obtain accesses to functional python 3, Rosetta software, PyRosetta interface, and have read through the tutorials of Rosetta and PyRosetta. Some basic Rosetta terminologies are used in this protocol, including 'pose', 'centroid', 'fullatom', 'mover', 'filter'. Their definitions can be found at https://www.rosettacommons.org/docs/wiki/rosetta_basics/RosettaEncyclopedia.

The following report uses a combination of bash commands, python scripts, xml scripts for RosettaScripts, and PyRosetta scripts. The documentation of bash commands can be found at https://www.gnu.org/software/bash/manual/bash.html#Shell-Commands. In this protocol, some basic bash commands are used to run the corresponding scripts. The documentation of python 3 can be found at https://docs.python.org/3/tutorial/. The resources, instructions for the installation, demonstration, and developments of Rosetta can be found at https://www.rosettacommons.org/. Rosetta licensing policy and software downloading instructions can be found at https://www.rosettacommons.org/software. It is highly recommended to obtain the license for Rosetta first, because the license is required for getting

access to the material provided on the RosettaCommons website. The tutorial and demos for using Rosetta can be obtained at https://www.rosettacommons.org/demos/latest/Home. The introductory resources for the usage of RosettaScripts can also be found in other reviews and articles (Chaudhury et al., 2010; Fleishman, Leaver-Fay, et al., 2011; Andrew Leaver-Fay et al., 2013; Leman et al., 2020). The resources, instructions for the installation, and demonstration of the PyRosetta interface can be found at http://www.pyrosetta.org/. The download and installation of PyRosetta can be found at http://www.pyrosetta.org/dow. The demos and tutorials of PyRosetta can be found at http://www.pyrosetta.org/tutorials. Jupyter Notebook is used during the analysis procedure, and the instructions for its download and usage can be found at https://jupyter-notebook.readthedocs.io/en/stable/index.html#. All demonstrative scripts and commands below can be executed on a personal portable computer (4 gigabyte random-access memory, Intel core i5 processor) with the Ubuntu operating system version 20.04, using the command line interface. Only one blueprint file will be generated, and less than 11 structures will be calculated at each step of the demonstration, which requires less than 50 megabytes of free disk space for smooth running during the whole procedure. The readers should adjust their preserved free disk space for program execution accordingly to prevent building-up of temporary files which may exhaust the disk spaces quickly. The python version used for the demonstration is 3.7.1. The Rosetta version used is v2020.25-dev61318 (The Rosetta version can be found on the name of the downloaded file). The PyRosetta version is PyRosetta-4 2020. PyRosetta is installed under a conda environment named 'pyrosetta' (this is the environment name used in this protocol, and readers should adjust accordingly), which is explained in the previously mentioned PyRosetta installation online resource. The authors recommend the readers to install the Jupyter Notebook under the same conda environment for the ease of use. All full versions of exemplary scripts and data sets are available at https://github.com/LAnAlchemist/blueprint_builder_demo. The linux operating system is not required, and the scripts and commands can be adapted to other operating systems. Please note that the syntaxes on Windows, MacOS, Unix, and Linux may be different, and misuse can lead to errors and/or abortion of the execution of the scripts and commands. Readers should make adjustments accordingly when following the exemplary commands. The first step of all demonstrations in the following examples are designed to change the current directory to the example directory. Most of the following commands can then be directly executed after removing the comments and line break signs if the demo folder structure is not changed by the reader. All commands and scripts are written in `Courier` font. Each line of commands starts with a larger sign (>) for clarity. A line break sign (\) is added at the end of the script or command if the script or command is not finished but broken into multiple lines for clarity. The type of the scripts or commands (e.g. python, bash, or xml) is noted with the number sign ('#') before the scripts or command body starts. Each command or script body is located between dashed lines for readability. When the example script content is explained, some parts of the scripts are omitted for clarity, which is noted with ellipsis signs (. . .). The explanations of each line of the scripts or commands are added after a number sign. If any action is triggered after running a script or a command for any step, the expected action and/or results will be noted and/or explained at the end of that step. Potential errors are noted in the 'Troubleshooting' section.

BASIC PROTOCOL 1: THE CONSTRUCTION OF BLUEPRINTS

Introduction

Following the funnel-shaped protein folding energy landscape hypothesis, previous studies demonstrated that the tertiary structure of a protein has strong dependence on the construction of its secondary structure elements (SSE) (Abkevich et al., 1996; Baker, 2019; Koga et al., 2012). Designing sequences to adapt the designed SSE patterns would favor the folding of the proteins to desired tertiary motifs driven by strongly funneled energy landscapes (Koga et al., 2012). Furthermore, Lin *et al.* extended the design rules by discovering the co-dependencies between the secondary structure features and the torsion angles of the loops linking them, and successfully designed proteins with specific secondary structure features (Lin et al., 2015). The blueprint pipeline is a method that allows users to pre-define each protein SSE and the local and non-local interactions between them, and design protein sequences which favors the realization of these SSEs and interactions. If successful, the designed protein sequence should adapt the designed SSEs and their interactions as constructed, which is expected as a result of adapting a sharp folding funnel and folding into the desired tertiary structure. These detailed secondary structure features with backbone torsion angle constraints used to direct protein folding into certain high-order structures are referred to as 'blueprints', and can be constructed as a text-based blueprint file (Figure 1). Rosetta BluePrintBDR ('BluePrintBDR' is a 'mover', if referred using the Rosetta terminology) was designed to read the blueprint file, and perform a stepwise Monte Carlo fragment assembly protocol to insert blueprint-biased protein fragments at each position of the designed protein (Bowie et al., 1994; A. Leaver-Fay et al., 2011; Marcos et al., 2017; Simons et al., 1997). An additional constraint file is sometimes supplied to specify certain structural geometry formations, such as to require hydrogen bonding formation through restraining the distance or angle between the atom pairs to the assigned values (see more explanations in Basic Protocol 1 Build blueprints from scratch). Both the blueprint and the constraint files are text-based files, and can be modified easily with any text editor, which gives users a large degree of freedom during the protein design process.

[*Place figure 1 near here.]

An example of a simple blueprint file is shown in Figure 1. Each line of the blueprint represents the construction of one residue. A tetra-peptide can be built from this blueprint. The first column on the left is a number, a non-zero residue number of the part from the input protein that will be kept and used as the building point, or 0 to mark new residues to be appended. The second column from the left is a one-letter amino acid code, which encodes the amino acid identity to be used for centroid-level structure building at that specific residue. Commonly, valine and/or alanine is chosen to represent an average side chain occupation. The primary sequence locations of the special amino acids such as glycines, prolines, and cysteines can be assigned at this stage if their locations are designed. The third column from the left defines the expected secondary structure feature of the residues (Lin et al., 2015; Wintjens et al., 1996). The first or the only letter in the third column defines the secondary structure feature of the corresponding residue, and is chosen from 'L', 'E', or 'H', which represents loop, extended strand, and helix, respectively. Sometimes, a second letter is added to the third column, which is a finer category representing the ABEGO bin of the assigned residue. The ABEGO logos, 'A', 'B', 'E', 'G', and 'O', represent certain blocks of backbone phi-psi angle combinations (Lin et al., 2015; Wintjens et al., 1996). Detailed definitions of the ABEGO bins can be found at (Lin et al., 2015).

The last column indicates whether the residue is going to be rebuilt (indicated by 'R') or required for keeping the current position (indicated by '.').

Commonly, two methods are used for generating blueprints. The first method is to build a blueprint file from scratch based on the meticulous analyses of the protein structures to be designed (see Basic Protocol 1). The first method will be discussed in more detail with the example of building a β-barrel. The second method is to build blueprints based on the structures of the existing proteins (see Alternative Protocol 1, (Huang et al., 2011)). This method can be useful if the users want to perform engineering, idealization, and redesign on the existing proteins of interest. PyRosetta can be directly used to generate corresponding blueprint files for any .pdb file through obtaining the knowledge of the secondary structural feature and torsional angle distribution for each residue. The use of a python script is provided as an example.

Basic Protocol 1: Build blueprints from scratch

Required resources

A computational system installed with a python script interpreter and PyRosetta interface.

Sample data

The exemplary input and output files generated in this section can be found at:

https://github.com/LAnAlchemist/blueprint_builder_demo/tree/master/demo1.1

Analysis of the protein folds

Building blueprints for protein folds requires a systematic understanding of the organization of the desired folds in the first place. Earlier studies demonstrated that the basic structural characteristic parameters of β-barrel proteins are the number of strands (n) and the shear number (S), which corresponds to the stagger of the closed β sheet as the relative position of the same strand can be displaced by following the specific backbone hydrogen bond pattern ((Murzin et al., 1994a, 1994b), Figure 2 a). Because the extended strand has neighboring side chains facing two sides, the antiparallel strands of a β-barrel has the feature that the residues with same backbone hydrogen direction also have their side chains facing the same side (i.e. inward or outward-facing of the β-barrel). The residues with the same side chain direction and connected by backbone hydrogen bonds are referred to as a Cβ-strip (Figure 2 b). The number of Cβ-strips is the half of the number of S (Figure 2 c). The radius of the β-barrel is determined by n, S, and the distance between neighboring strands (D, Figure 2 c). Analysis of native β-barrel structures suggested that the soluble β-barrels usually have the n of 8 or 10, and S of 8, 10, or 12. A β-barrel of n = 8 and S = 10 was built with the aim of designing soluble and stable β-barrel proteins which are big enough to accommodate small molecule ligands (Dou et al., 2018).

[*Place figure 2 near here.]

Generation of the 2D map and its corresponding blueprint for β-barrel structure building

To build blueprints from scratch, a 2D map is usually built for a specific fold of protein for design. It is usually recommended to first analyze structurally related native or engineered proteins, and use the knowledge learnt to help the construction of the 2D map. While it bears

the name of a 'map', it can adapt any forms that can be understood by the users and translated into a blueprint file. The 2D map needs to include a suite of information on the protein to be designed, including the total protein residue number, the length of each secondary structure element, the backbone hydrogen bonds (i.e. the hydrogen bond donor atom, residue and the hydrogen bond acceptor atom, residue), and the backbone torsion angle bin of each residue (Lin et al., 2015).

Based on the above-mentioned analysis of native β-barrel structures, a 2D map (Figure 2 c) was generated to build a specific de novo β-barrel. This example was prepared for a β-barrel with eight strands (n = 8) and shear number 10 (S = 10). The residue numbers that compose each secondary structure element and interact through backbone hydrogen bond pairs can be developed from this 2D map. The length of each β-strand was chosen to satisfy the pre-defined registry shifts following the example in (Dou et al., 2018). The length and torsion angle of the loop regions between each strand were also pre-defined (Figure 2 d). As a result, a string representing the secondary structure assignment for the example β-barrel fold became 'L1E9L3E13L2E10L3E13L2E8L4E11L2E8L3E10L1', i.e. a nine-residue strand followed by a three-residue loop, a thirteen-residue strand, etc. More detailed structural features implied in the 2D map were indicated during the process of constructing a blueprint file. For this 2D map to be understood by Rosetta, we need to convert the 2D map into a blueprint file (Figure 1). With the blueprint format, the structure to be folded will be understood in residue-level, each describing the intended three-state secondary structure type ('H', 'E', or 'L') and/or more specific backbone torsion angle bin (ABEGO bin). All residues composing strands were indicated with 'E', residues forming the N terminal helix were written with 'H', and the other loop residues were assigned with 'L' in the blueprint file. The length of each continuous secondary structure segment followed that assigned by the 2D map. The rules used for defining the ABEGO bin for each residue are listed here as an example. 1) Regular extended structure-residues composing the β strands ('E') are with ABEGO label 'B', 2) Some glycine residues are placed in the middle of long strands ('E') to release Cβ-strip strains (referred to as 'glycine kinks', with ABEGO label 'E', 3) One to two β-bulge residues ('E') are placed at the end of assigned strands to also relieve the strain and drive the curve of the β-sheet, with ABEGO label 'A' , and 4) Loop residues ('L') between the β-strands designed to form specific β-turn types are with ABEGO label sequences 'AA', 'AAG', or 'AAAG' depending on their positions within the 2D map (Dou et al., 2018). Further specification can additionally be applied using the constraints that will be described in the next section.

A python script was generated for the preparation of the blueprint and the constraint file at the same time and, the usage of the script will be demonstrated in the following step.

Generation of a constraint file for β-barrel

While a blueprint file tells Rosetta the required fragment feature for each position, a constraint file will provide additional information to Rosetta to bias the sampling of conformations. The constraint files are not required, but recommended if building complicated proteins, such as β-barrels, to improve folding efficiency.

The constraint file for building the β-barrel defines its hydrogen bond registry between each Cβ-strip (Figure 2 b & c). Other distance and angle constraints can be defined in similar fashion. Each line of constraint follows the format of 'type of constraint', 'applied atom1', 'residue

number of atom1', 'applied atom2', 'residue number of atom2', ... , 'function type used for the constraint', 'required distance or angle', and 'deviation', and connected by space. To restrain a backbone hydrogen bond, both atom pair distance and bond angle are defined following the previous studies (Figure 3) (Dou et al., 2018). The first line describes the distance requirement between the heavy atom of the acceptor hydrogen, and the heavy atom of the donor, which is N17 and O21, respectively. The function used is a harmonic function, with a requirement distance of 3 Å, and 0.5 Å deviation. The second line defines the angle between three atoms, N17-H17-O21, which is one of the components to describe the hydrogen bond. The function used is circular harmonic, with a requirement of radian 3.1, and 0.3 degree deviation. More examples of constraints can be found at https://www.rosettacommons.org/docs/wiki/rosetta_basics/file_types/constraint-file.

[*Place figure 3 near here.]

The stepwise commands to generate blueprint and constraint files are described below:

1. Open your command line interface.
2. (Optional) Make a new directory as your working directory and move the demo folders into it. It is suggested that the same directory be used to contain all the files for constructing the protein.

   ```
   #bash script-------------------------------------------------
   --

   >mkdir /path_to_document/my_beta_barrel_folder #You may name
   this directory by your favorite name

   >mv /path_to_where_the_demo_folders_are/* \
   /path_to_document/my_beta_barrel_folder #Move the extracted
   downloaded demo folders and all files in the folders to the
   newly created directory. '*' represents all files and folders
   under the indicated directory. In this case, they should be
   the extracted demo folders and files.

   #-------------------------------------------------------------
   --
   ```

   Expected results: A new directory, `/path_to_document/my_beta_barrel_folder/`, was created. Then the downloaded demo files and folders have been moved into that directory.

3. Move to your working directory. It is recommended to store all generated files in this directory.

   ```
   #bash script-------------------------------------------------
   --

   >cd /path_to_document/my_beta_barrel_folder/demo1.1 #move to
   your working directory
   ```

```
#-------------------------------------------------------------
--
```

Expected results: Your current path is changed to folder `demo1.1/`.

4. Use the pre-developed python script to convert a 2D map into a blueprint file.

```
#bash script-----------------------------------------------
--

>python ./scripts/gen_bp_cst.py \
"L1E9L3E13L2E10L3E13L2E8L4E11L2E8L3E10L1" \
"E1.1;E2.5;E3.8;E4.5,7;E6.5;E8.5" bp cst #The python script
'gen_bp_cst.py' takes the string of listed secondary
structures to be built, followed by a string indicating the
location of pre-defined glycine kinks and the names of the
output blueprint and constraint files to be generated, 'bp'
and 'cst', respectively.

#-------------------------------------------------------------
--
```

Expected results: A blueprint file (`./bp`) and a constraint file (`./cst`) are created.

5. (Optional) Check your resulting blueprint file

```
#bash script-----------------------------------------------
--

>head ./bp #check the top several lines of the blueprint file

#-------------------------------------------------------------
--
```

Expected results: The top few lines of the blueprint fine is printed on your terminal screen.

Alternate Protocol 1: Build blueprints based on existing protein .pdb files

Required resources

A Computational system installed with a python script interpreter, and PyRosetta.

Sample data

The exemplary input and output files in this section can be found at: https://github.com/LAnAlchemist/blueprint_builder_demo/tree/master/demo1.2

Generate a blueprint file based on a native .pdb file

The stepwise commands are described below:

1. Open your command line interface, (optional, create your working directory) and move to your working directory following the previously described steps.

```
#bash script-------------------------------------------------
--

>cd /path_to_document/my_beta_barrel_folder/demo1.2 #Move to
folder demo1.2

#-----------------------------------------------------------
--
```

2. Create the blueprint file based on a .pdb file of your choice:

```
#bash script-------------------------------------------------
--

>conda activate pyrosetta #(This step may require adjustments
depending on the installation steps of the users.) This step
activates a conda environment, 'pyrosetta', under which
pyrosetta and related modules are installed. The authors named
their conda environment 'pyrosetta', and the users should
change the name accordingly.

>python ./scripts/pyrosetta_pdb_to_bp.py ./4rlc.pdb #The
demonstrative .pdb file, 4rlc.pdb, can be replaced with your
file of choice. The argument following the .py scripts
specifies the location of the chosen .pdb file to calculate
the blueprint for.

#-----------------------------------------------------------
--
```

Expected results: Information about calling of PyRosetta is printed on the screen. After execution, you should see the creation of a new blueprint file in your working directory. For this particular case, '`4rlc_bp`' is created.

BASIC PROTOCOL 2: DE NOVO PROTEIN DESIGN BASED ON THE BLUEPRINTS

Introduction

With the blueprint file and the corresponding constraint files in hand, we can start to use Rosetta to build the backbone of the β-barrel, and perform sequence design on these backbones. This backbone and sequence design procedure is largely shared in other Rosetta protein design protocols. Many of the RosettaScripts used in the demonstration can be directly adapted by pipelines not using the BluePrintBDR. We will first build the backbone of the protein using the BluePrintBDR through a stepwise Monte Carlo fragment insertion of protein oligomer at each residue ((Bowie et al., 1994; Simons et al., 1999b; Simons et al., 1997), see COMMENTARY for other backbone building methods). A filter step will be followed to select backbones with satisfactory quality. Next, side chains will be designed based on these protein backbones,

followed by another round of filtering to select desired sequences. The output sequences can be selected for further wet laboratory verification, which is beyond the content of this protocol, and will not be discussed in this protocol.

In the backbone building step, the BluePrintBDR will first insert protein fragments at each position of the to-be-built protein residue starting from a position in the input protein based on the requirement of the blueprint file (Bowie et al., 1994; Simons et al., 1999b; Simons et al., 1997). While we call the generated structures 'backbones', they are usually proteins constructed with polyvaline or polyalanine sequence. Their backbone conformation will be used and generally preserved during the sequence design step, and determine the overall fold of the designed protein. The constraint file is optional, and if it is provided, the refinement of the designed protein is performed with the consideration of the constraint requirement. The constraints can efficiently restrict the conformational space to be explored during the structure building process, thus they are often applied for building complicated proteins. During building of the β-barrel, constraints were applied to enforce the formation of hydrogen bonds with accurate registry shift between the strands as intended. After Monte Carlo-based fragment assembly, the structure is minimized in centroid mode. The constraints are applied both during the Monte Carlo procedure and energy minimization steps. The final model is converted into fullatom mode structure for scoring (for further explanation on centroid mode and fullatom mode, please check https://www.rosettacommons.org/demos/latest/tutorials/full_atom_vs_centroid/fullatom_centroid). The specific script lines for above-mentioned key steps are commented in the sample xml script provided at the end of this section.

Followed by the backbone building, a filter step is usually added to select backbones with satisfactory quality. Because the following step, sequence design, is usually the most resource-consuming step, only backbones with certain qualities are used to save computational resources. In general, backbone structures with high quality will also result in better sequence design results. In the demonstration, a Jupyter Notebook is used at this step for its convenience for data visualization (Kluyver et al., 2016). Usually, score terms and filters including 'vdw', 'rama', 'omega', 'hb_lr_bb', 'hb_sr_bb', etc, are checked. Additional explanations on different score terms can be found in the demonstrative Jupyter Notebook and at https://www.rosettacommons.org/docs/wiki/rosetta_basics/scoring/centroid-score-terms. It is recommended to take the protein backbones scored by top percentage based on the experience and intentions of the users, because it is impossible to strictly define hard score cutoff values that are universally applicable for different protein design situations.

After selecting the protein backbones, these backbones are used for a few rounds of sequence design on the fixed backbone and structure minimization on the fixed sequence, which are referred to as the 'sequence design' steps. Sometimes, the designers will be able to identify locations that need to be conserved for selected amino acids. A 'resfile' is used to identify these locations and restrict the amino acid choices during design. In the provided example, some glycine kinks were identified in the resfile to release the inner tension of each Cβ-strip. Also, in general, a layer design approach is used in the sequence design step. All residues can be classified into 'core', 'boundary', or 'surface' type based on the number of their geometric-neighboring residues, or their solvent accessible surface area. In the demonstration, all residues were also classified into different SSE, i.e. loop, strand, or helix, based on the DSSP definition of

secondary structures (Kabsch et al., 1983). The combination of these two classifications allocate each residue into different layers, such as 'coreH' (i.e. a helical residue in the core region) or 'surfE' (i.e. a strand residue on the surface) (Dou et al., 2018). Residues in each layer can be assigned with different design operations. For example, the user can avoid the use of hydrophobic residues for surface residues and prefer proline at the end of a helix. Special locations with specific amino acid types (often glycine, cysteine, or proline) are mostly pre-defined, disallowed to be designed, and only allowed for packing.

Required resources

A computational system installed with a python script interpreter, PyRosetta, Rosetta, and Jupyter Notebook. Computing in computer clusters are recommended for computationally heavy steps, which are noted in stepwise instructions.

A working directory containing the blueprint file and the constraint file created from the previous steps.

Sample data

The exemplary input and output files generated for '***Protein backbone assembly using the BluePrintBDR'*** *and* ***'Filtering of the backbones designed from the BluePrintBDR'*** in this section can be found at:
https://github.com/LAnAlchemist/blueprint_builder_demo/tree/master/demo_bb

The exemplary input and output files generated for '***Sequence design on the selected backbone structures***' *and* '***Filtering of the designed proteins***' in this section can be found at:
https://github.com/LAnAlchemist/blueprint_builder_demo/tree/master/demo_seq

Protein backbone assembly using the BluePrintBDR

1. Open your command line interface, (optional, create your working directory) and move to your working directory following the previously introduced steps.

   ```
   #bash script-------------------------------------------------
   --

   >cd /path_to_document/my_beta_barrel_folder/demo_bb #move to
   your working directory and folder, demo_bb/.

   #-----------------------------------------------------------
   --
   ```

2. Run the following commands to perform backbone assembly following protocol, bb_gen.xml. An explanation of part of the demonstrated xml file is provided in the coming section. While the below demonstration directly runs the backbone design jobs on the personal computational system, it is recommended to submit the jobs to computer clusters, especially if many structures are to be calculated. Readers should inquire about their local computing resources for submission method.

   ```
   #bash script-------------------------------------------------
   --
   ```

```
>/path_to_your_Rosetta/rosetta/main/source/bin/rosetta_scripts
.default.linuxgccrelease \ #Call your rosetta script
-parser:protocol ./bb_gen.xml \ #The flag '-parser:protocol'
followed by an xml file instructs the RosettaScripts to
execute the objects listed in the xml protocol section.
-s ./input.pdb \ #the supplied input pdb file followed by the
flag '-s' will be used as an initial position of a protein
fragment to build the protein with the BluePrintBDR.
-parser:script_vars bp=./bp cst=./cst \
weights=./fldsgn_cen_omega02.wts \ #The flag '-
parser:script_vars' supplies the files which are called by the
xml protocol during the execution of the RosettaScripts. Those
variants are quoted through citing variant names using '%%'
signs. More explanations for calling of the external files are
provided in the coming section, together with the explanation
of the xml file.
-out:pdb:path ./result \ #An optional flag, The flag '-
out:pdb:path' supplies the location of the directory where you
want to store the resulting .pdb file. The default path to the
resulting pdb file is the current working directory, if not
specified.
-out:score:path ./result \ #An optional flag, The flag '-
out:score:path' supplies the location of the directory where
you want to store the resulting score file. The default path
to the resulting score file is the current working directory,
if not specified.
-out:suffix _demo \ #An optional but highly recommended flag,
the flag '-out:suffix' supplies a suffix after all resulting
pdb, which helps identifying each experiment results if
multiple rounds of protein constructions were performed. Other
than the suffix option, a prefix can be added in a similar way
using the flag '-out:prefix your_prefix_'.
-nstruct 10 \ #The flag '-nstruct' specifies the number of
calculation rounds to be performed with the xml protocol. This
leads to the production of 10 designed structures at maximum
if they all passed the filters indicated in the xml protocol.
#-------------------------------------------------------------
-
```

Expected results: Information on the execution of the RosettaScripts is printed on the screen. This step may take around 10 to 30 min for each structure to finish computing, depending on the specific protein folds to be built. After the running is completed, you should see the creation of the de novo protein .pdb files and its corresponding score file in the specified result directory. You may visually check the .pdb files using PyMOL (The PyMOL Molecular Graphics System, Schrödinger, LLC). In this particular demo, ten .pdb files and a score file are created in the directory, '`./result`'.

To explain how the BluePrintBDR works, a part of the backbone generation xml file, 'bb_gen.xml', is selected and explained below. It is recommended that the readers first read the protocol part (i.e. the part of the xml script blocked with lines <PROTOCOLS>...<PROTOCOLS/>) and track the operation names in the corresponding parts of the xml script (such as <MOVERS> and <FILTERS> blocks) to understand their roles when called. A brief introduction on the demonstrated xml script is added in the comments section for each line. Additional documentations on RosettaScripts can be found at https://www.rosettacommons.org/docs/wiki/Home.

```
#RosettaScripts, in xml format-------------------------------------
-

<ROSETTASCRIPTS> #Here is the start of the whole Rosetta Script.

<SCOREFXNS> #Here is the start of the section to define scoring
functions.

     <ScoreFunction name="SFXN1" weights="%%weights%%" > #The
supplied weights file, 'weights=./fldsgn_cen_omega02.wts', is cited
here.

     ...

     </ScoreFunction>

...

</SCOREFXNS> #Here is the end of the score function section.

<FILTERS> #Here is the start of the filter section.

     ...

     <ScoreType name="vdw" scorefxn="SFXN1" score_type="vdw"
threshold="1000000" /> #This filter with score type 'vdw' will
calculate the van der Waals energy of the generated protein
structure and reject the structure if the energy is higher than the
provided threshold. A relatively high threshold is set here as an
example so probably no structure is rejected during the running of
this filter.

     ...

<FILTERS/> #Here is the end of the filter section.

...

<MOVERS> #Here is the start of the mover section.

...
```

<Dssp name="dssp"/> #This mover performs secondary structure assignment for each residue of the protein using the DSSP algorithm (Kabsch et al., 1983).

<SwitchResidueTypeSetMover name="fullatom" set="fa_standard"/> #This mover converts the representation of the protein model into fullatom mode.

<SwitchResidueTypeSetMover name="cent" set="centroid"/> #This mover converts the representation of the protein model into centroid mode. In the centroid mode, only the backbone atoms retain their original features and the side chain atoms are represented by a single virtual atom for simplification.

...

<BluePrintBDR name="bdr1" scorefxn="SFXN1" use_abego_bias="1" blueprint="%%bp%%" constraint_file="%%cst%%"/> #This defines the blueprint builder mover with specifications. The scoring function defined as 'SFXN1' is used during the execution of the building. The flag 'use_abego_bias' is set to 'true' ('1') to bias fragment picking with the ABEGO bins of the corresponding residue positions of the protein as assigned in the blueprint file. The sign '%%' cites the external variants provided following the flag '-parser:script_vars' in the RosettaScripts execution command. The provided blueprint and constraint files are cited here.

<ConstraintSetMover name="addcst1" add_constraints="1" cst_file="%%cst%%"/> #This mover adds the provided constraint to the protein structure processed during RosettaScripts execution.

<MinMover name="min1" scorefxn="SFXN1" chi="1" bb="1" type="dfpmin_armijo_nonmonotone_atol" tolerance="0.0001"/> #The mover defined by this line performs minimization of the built protein structure. In this example, the degrees of freedom allowed to be minimized include the chi angles and backbone torsion angles as indicated by '1' for both 'chi' and 'bb' flags. The type of the minimization algorithm to be used is defined as 'dfpmin_armijo_nonmonotone_atol' with absolute tolerance '0.0001'. The 'tolerance' sets the requirement for minimization convergence and the smaller number gives higher requirement for convergence.

...

<ParsedProtocol name="cenmin1" > #This block includes the specific steps of switching the protein representation to centroid mode for minimization and switching back to fullatom mode. Protein structure in fullatom mode will later be used for scoring, filtering, and construction in the final result .pdb file in the

<PROTOCOLS> block. This section contains four individual movers, which are included between <ParsedProtocol> ...<ParsedProtocol/>.

<Add mover_name="cent" /> #See above for explanation.

<Add mover_name="addcst1" /> #The provided constraints should be added before the minimization.

<Add mover_name="min1" /> #To perform minimization of the generated structure.

<Add mover_name="fullatom" /> #See above for explanation

</ParsedProtocol>

<ParsedProtocol name="bdr1ss" > #This block indicates the specific steps to set up the BluePrintBDR. This section includes three individual movers, “bdr1”, ”cenmin1”, and ”dssp” , which are included between <ParsedProtocol> ...<ParsedProtocol/>.

<Add mover_name="bdr1" /> #Calls the mover, BluePrintBDR.

<Add mover_name="cenmin1" /> #This converts the structure into a centroid mode for protein structure building and minimization. Centroid mode representation allows the energy calculation during fragment assembly to be done in lower resolution.

<Add mover_name="dssp" /> #See above for explanation

</ParsedProtocol>

</MOVERS> #Here is the end of the mover section.

<PROTOCOLS> #Here is the start of the protocol section

<Add mover_name="bdr1ss" /> #Calls the mover, BluePrintBDR

...

<Add filter_name="vdw" /> #Applies the score type filter to calculate the ‘vdw’ energy, which we will use for structure filtering in the next step.

...

<Add mover_name="fullatom" /> #This mover converts the designed structure into fullatom mode.

...

</PROTOCOLS> #Here is the end of the protocol section

</ROSETTASCRIPTS> #Here is the end of the whole RosettaScript.

```
#-----------------------------------------------------------------
-
```

Filtering of the backbones designed from the BluePrintBDR

3. Open your command line interface, (optional, create your working directory) and move to your working directory following the previously introduced steps.

```
#bash script-----------------------------------------------------
--

>cd /path_to_document/my_beta_barrel_folder/demo_bb #move to
your working directory, change to folder, demo_bb/.

#-----------------------------------------------------------------
--
```

4. Open Jupyter Notebook. If the reader had the Jupyter Notebook installed under a conda environment, as suggested in the Introduction, the reader will need to activate that conda environment following the previously introduced command before executing this step.

```
#bash script-----------------------------------------------------
--

>jupyter notebook # open Jupyter Notebook application

#-----------------------------------------------------------------
--
```

Expected results: A new html-based window pops out, which displays the contents of your current directory.

5. Click to open the Jupyter Notebook, '`./scripts/pick_bb.ipynb`', follow the instructions in the notebook to complete the backbone quality analysis. Stepwise explanations are provided in the notebook before each cell, thus will no longer be repeated in the main text of this protocol.

Expected results: A new folder (`./sel_for_seq_design/`) containing all selected backbone structures is created.

Sequence design on the selected backbone structures

6. Open your command line interface, (optional, create your working directory) and move to your working directory following the previously introduced steps.

```
#bash script-----------------------------------------------------
--

>cd /path_to_document/my_beta_barrel_folder/demo_seq #move to
your working directory
```

```
#-------------------------------------------------------------
--
```

7. (Optional but recommended, not demonstrated here) Make another directory based on the fold to be generated and move or copy the corresponding blueprint file and constraint file into it. This step will help data management if you are working with multiple protein folds to be built based on different 2D maps at the same time. In the demonstration, because structures for only one blueprint is built, the blueprint and the constraint files are directly copied to `demo_seq/` for further steps.

8. Run the following commands to generate the resfile and the sequence design xml file for downstream sequence design. The resfile contains pre-defined protein fold-specific amino acid type requirements for certain locations. In the example of the β-barrel construction, the resfile was generated to define: 1) the location of glycine kinks to release Gβ-strip tension; 2) the amino acid type requirements for the β-turns; 3) the location of tryptophan corners (Dou et al., 2018).

```
#bash script-------------------------------------------------
--

>python ./scripts/gen_resfile_xml_from_bp.py ./bp \
./bb_design.resfile ./bb_design.xml #The
'gen_resfile_xml_from_bp.py' file is the python script that
creates the resfile and the sequence design xml file using the
blueprint file generated in Basic Protocol 1. The script is
written to create a new xml file compatible with given inputs
by modifying the template xml file, 'template_bb_design.xml'.
Thus it is recommended that the python script,
'template_bb_design.xml' file, and the input blueprint file be
stored under the parent directory to simplify the executing
procedure. The first argument following the python script
specifies the location of the blueprint file. The second
argument specifies the desired location and name of the output
resfile. The last argument specifies the desired location and
name of the output sequence design xml file.

#-------------------------------------------------------------
--
```

Expected results: The resfile and sequence design .xml file are created in the specified directory.

9. Run the following commands to generate the running scripts for execution of sequence design using Rosetta. One command line is designed to be generated for each selected backbone structure generated from the previous backbone generation step.

```
#bash script-------------------------------------------------
--
```

```
>python ./scripts/seq_cmd_generator.py \
/path_to_your_Rosetta/rosetta/main/source/bin/rosetta_scripts.
default.linuxgccrelease ./bb_design.xml ./bb_design.resfile \
/path_to_document/my_beta_barrel_folder/demo_bb/sel_for_seq_de
sign #seq_cmd_generator.py is the python script that creates a
file containing all commands to perform sequence design on
each selected backbone structure.
#-------------------------------------------------------------
--
```

Expected results: A '`seq_design_cmd`' file is created, which contains the same number of command lines as the number of the selected backbones. Each line is a command line of using RosettaScripts for performing sequence design on a particular backbone structure.

10. Run the following commands to perform sequence design. In this step, the commands generated in step 9, which were set up to run the 'bb_design.xml' script with its inputs, are executed. Each structure may take 20 ~ 60 min to calculate. While the below demonstration directly runs the sequence design jobs on the current personal computational system, it is highly recommended to submit the jobs to computer clusters, especially if many structures were to be designed.

```
#bash script-------------------------------------------------
--
>bash ./seq_design_cmd #Execute the sequence design commands.
#--------------------------------------------------------------
--
```

Expected results: The designed proteins and a corresponding score file are created in the specified directory, `./seq_result/`. You may visually check the pdb files using PyMOL (The PyMOL Molecular Graphics System, Version 1.2r3pre, Schrödinger, LLC).

To explain the sequence design procedure, a part of the sequence design xml file, '`bb_design.xml`', is explained below. The same reading method is recommended as mentioned before.

```
#RosettaScripts, in xml format-------------------------------------
-

<ROSETTASCRIPTS> #Here is the start of the whole Rosetta Script.

    ...

<RESIDUE_SELECTORS> #Here is the start of the residue selectors
block, which defines different subsets of residues to work on.

     <Layer name="coreRes" select_core="true"
use_sidechain_neighbors="true" core_cutoff="2.1"
```

```
surface_cutoff="1.0"/> #This residue selector selects the core
residues. The detection is based on counting the number of
contacting neighboring side chains of that residue. In this
particular case, if a residue has more than 2.1 contacting residues,
it will be counted as a core residue. If a residue has less than 1.0
contacting residue, it will be counted as a surface residue.

     ...
     <SecondaryStructure name="helix" ss="H"
pose_secstruct="LLHHHHLLEEEEEEEELLLEEEEEEEEEEEELLEEEEEEEEELLLEEEE
EEEEEEEELLEEEEEELLLLEEEEEEEEEELLEEEEEEELLLEEEEEEEEEL"/> #This
residue selector selects all helical residues based on their
individual secondary structure state assignments. For this example,
the states of each residue in the input backbone structure were
assigned according to the DSSP algorithm (Kabsch,W. and Sander,C.
(1983) Biopolymers 22, 2577-2637).

     ...

     <And name="coreH" selectors="coreRes,helix,not_resfile_res" />
#This residue selector is an 'AND' logical selector, which takes the
overlapping residues of the indicated individual selectors. This
particular selector selects core residues which also have 'H'
secondary structure states, but are not specified in the provided
resfile.

</RESIDUE_SELECTORS> #Here is the end of the residue selectors
block.

<TASKOPERATIONS> #Here is the start of the block defining task
operations.

     ...
     <ReadResfile name="resfile" filename="%%resfile%%" /> #This
task operation reads the resfile provided by the 'resfile' variable
in the execution command and correspondingly defines design rules
for specified residue positions.
     <OperateOnResidueSubset name="design_helixCore_AA"
selector="coreH" >
     <RestrictAbsentCanonicalAASRLT
aas="AFILVM"/></OperateOnResidueSubset> #This task operation defines
a task to perform on 'coreH' residues (see above) by restricting
them to be designed with selected amino acid types, 'AFILVM'.

     ...

</TASKOPERATIONS> #Here is the end of the task operations block.

<MOVERS> #Here is the start of the movers block.
```

```
      <FastDesign name="fdesign"
task_operations="resfile,design_helixCore_AA,...,design_loopSurf_AA"
scorefxn="beta" cst_file="%%cst_trp%%" ramp_down_constraints="1" />
#This Mover performs the major sequence design step. The
'task_operations' lists multiple tasks to define the overall
specifications of the sequence design steps to be performed. It will
logically merge all design rules defined by each task operation such
as 'resfile' and 'design_helixCore_AA' (see above for the
explanation) and construct a final map that controls the behavior of
each residue upon fast design application. For example, the final
map may define whether a residue is designable, and allowed amino
acid types if the residue is open for design. There are multiple
flags to define available options of fast design. In this example,
the energy function is defined by the 'scorefxn' and 'cst_file'
flags. The 'ramp_down_constraints' flag was set as true ('1') to
allow ramping down of the constraint weight.

      ...

</MOVERS>#Here is the end of the movers block.

...

<PROTOCOLS>#Here is the start of the protocol block.

      ...

      <Add mover_name="fdesign" /> #This calls the fast design
mover.

</PROTOCOLS>#Here is the end of the protocol block.

</ROSETTASCRIPTS> #Here is the end of the whole Rosetta Script.

#-----------------------------------------------------------------
-
```

Filtering of the designed proteins

11. Open your command line interface, (optional, create your working directory) and move to your working directory following the previously introduced steps.

```
#bash script-------------------------------------------------
--

>cd /path_to_document/my_beta_barrel_folder/demo_seq #Move to
the 'demo_seq/' directory under your main working directory

#-----------------------------------------------------------
--
```

12. Open Jupyter Notebook

```
#bash script-------------------------------------------------
--

>jupyter notebook

#-----------------------------------------------------------
--
```

Click to open the Jupyter Notebook, `'./scripts/SeqAnalysis.ipynb'`, and follow the instructions in the notebook to complete the protein sequence design quality analysis. Stepwise explanations are provided in the notebook before each command line, thus no longer repeated in the main text of this protocol. Other available filters and sequence quality control options are discussed in the 'Troubleshooting' section in the protocol.

COMMENTARY

Background Information

History of pipeline development

In early studies, sequence-based fragment assembly was used to perform ab initio protein structure prediction (Bowie et al., 1994; Simons et al., 1997). Its success demonstrated that understanding of the substructures will be extremely useful in recovering the whole protein tertiary structure (Simons et al., 1999a). With the success in protein structure prediction, the idea of reversing the structure prediction procedure for protein design was proposed. Designing sequences that adapt ideal motifs which are stabilized by local and distant interactions was proposed and tested, which was the early successes of de novo protein design (Kuhlman et al., 2003; Nauli et al., 2001). Following these early successes, a more general method to define the per-residue features of the protein that will be designed was developed (Koga et al., 2012; Lin et al., 2015) and applied to a variety of protein folds successfully (see Introduction), which became the widely used blueprint pipeline today.

Comparison to other methods

While the blueprint pipeline has proved itself powerful, it has the limitations of requiring a great understanding of the structures of to-be-designed proteins, which may be challenging if the targeting fold is structurally complicated. Additionally, the BluePrintBDR cannot change the length of the designed proteins, nor can it introduce fold-level diversification into the protein built. Lastly, the result of the fixed-backbone sequence design step can be limited by the quality of the backbones generated using the blueprint. The Fleishman laboratory developed methods reaching rich structure diversity through combining native protein fragments (Lapidoth et al., 2015), and successfully designed enzymes with comparable characteristics to native enzymes (Netzer et al., 2018). Other recently developed methods use machine learning to directly generate protein sequences for desired or novel protein folds (Anand et al., 2020; Anishchenko et al., 2020), which at the same time can provide proteins with great structural diversity. Other methods include SEWING (Jacobs et al., 2016), junction fusion protein creation (Brunette et al., 2020), and loop-helix-loop unit combinatorial sampling (Pan et al., 2020).

Critical Parameters

Structural parameters defined by the blueprint designed from scratch

The BluePrintBDR only builds the structure as asked for by the provided blueprint. The fragments used for assembly are picked with bias to the assigned features in the blueprint. If a user is aiming to build a fold from scratch, a meticulous analysis on the corresponding or related folds is strongly recommended prior to generating a blueprint. Every detail in the input blueprint and constraint file can affect the output structure and will decide the feasibility of the structure to be folded as defined by the user. For example, the location of β-bulges is critical to determine the folding of β-barrel and NTF2-like proteins (Dou et al., 2018; Marcos et al., 2017; Marcos et al., 2018). Constructing multiple 2D maps or blueprints with different options in structural parameters such as the length of the secondary structure segments or secondary structure states may aid in exploring the parameter space. Please refer to the 'Troubleshooting' section for some suggestions on how to improve the quality of the blueprints.

Constraint parameters

Constraints can optionally be constructed and assigned to restrict the folding of the structures during the blueprint building process. Each constraint specifies a function type such as a harmonic function or a circular harmonic function and its related parameters to define a potential. The parameters assigned for each constraint can change the minimum or the steepness of the potential and thus reflect the target value to constrain to or how strongly the user wants the restriction to be applied. The minimum value can be decided based on biophysical studies or experiences of the user (i.e. average value observed for the distance between two atom types). For the standard deviations of harmonic or circular harmonic functions, we provided the parameters that were used in (Dou et al., 2018) but it can be changed with caution depending on the experiences or intentions of the users. The smaller the standard deviation, the stronger the constraint will be forced, but such constraints can sometimes interrupt energy minimization or cause irregularities in the protein structure by relatively underestimating the raw scoring function. Details on the constraint types and function types applicable by Rosetta can be found at https://www.rosettacommons.org/docs/wiki/rosetta_basics/file_types/constraint-file.

Number of structures to generate

When a centroid-level structure is generated for a given blueprint, a random factor exists when a fragment or a position to be inserted is picked. Different fragments can be selected every time while satisfying the selection condition provided by the blueprint. A trial fragment insertion is accepted based on the Metropolis criterion which also involves randomness. Therefore, even when one blueprint file is given as input, if multiple structures are generated with the number parameter assigned after the flag '`-nstruct`', structural diversity in other dimensions can be observed. Statistically, we may expect the structures to be distributed more finely when more structures are generated while maintaining the blueprint definition. This resolution of structural difference can affect the downstream sequence design results and hence the number of centroid-level structures to generate can be a critical parameter for the quality of final design outputs. In another case, when the success ratio is lower than expectation after applying the backbone structure filtering criteria, the user might want to generate more structures with the

same blueprint builder by increasing the number parameter. Hundreds of centroid-level structures are usually generated with the BluePrintBDR for well-studied blueprints (Dou et al., 2018).

Troubleshooting

General steps for troubleshooting when there is error during program execution

Observation: The execution is aborted, an 'ERROR' code pops up, or no expected results are found after execution.

When the execution of a script is aborted abruptly and if there is an execution log file, the user should go back and trace the log. The logs are usually the record of computational processes printed as words during the computational execution. The log can be printed out on the screen during the execution of the scripts or saved as a file. You should find the lines marked with 'ERROR' and check its content. The error line usually tells the specific script line number/function name of where the error occurs. Suggested methods to fix the error are sometimes described following the error codes. Common errors for new users include spelling errors, indentation errors, input file formatting errors, and path errors, which are recommended to be checked first. If the user does not understand the meaning of the error code, we encourage the users to first search the error code online for explanations. If an error occurs when the user is executing a Rosetta program, a 'ROSETTACRASH.log' file will be created, which indicates the specific error location. The user should troubleshoot with previously mentioned general methods first. If the error remains or too little information is provided by the log file, the user is encouraged to search for the solution and/or ask for assistance at https://www.rosettacommons.org/.

Blueprints improvements

Observation: The score distribution of the structures is generally unsatisfactory or successful rate of generating structures passing the recommended score threshold is low. The generated structures do not comply with the requirements indicated by the input blueprint and constraint files.

Some common score terms can be examined first to find potential issues with the generated structures. High 'vdw' score indicates the structure may have steric clash and high 'omega' and 'cen_rama' scores suggest the structure may have non-ideal geometry. Low 'ss_pred' score may suggest the resulting protein has low chance in satisfying the required secondary structure features.

One potential reason of the structures being unfavorably scored could be that the definition and requirements set up with the blueprint and constraint file are not compatible with ideal structure formation. For example, if a definition of the ABEGO bins of your partial input is hard to be find in the native protein structure database, high quality fragments will not be picked, and the generated structure can end up having clashed and unstructured parts. The general strategy to solve these issues are to identify 'badly defined regions' and optimize their blueprint (and/or constraint) definitions. Users are suggested to analyze the SSE and ABEGO bins of the output structures to check if they fit the original blueprint requirements (see Alternative

Protocol 1). Also, it is suggested that users analyze the atom-pair angles and distances of the resulting structures to check if they meet the original constraint requirements, if a constraint file is provided. If there are residues that fail the requirements, this may indicate the original residue requirements are unrealistic, difficult to satisfy, or even self-contradicting. The users may make changes on those residues to screen for more reasonable blueprint requirements.

It is also suggested to perform per-residue analysis to check the scores for each residue to find poorly performing regions. Based on the score term that results in unfavorable scores, users may conclude which characteristic of each residue needs improvements. Common issues include formation of clashes, unideal geometries, and unsatisfied buried polar functional groups. These issues may suggest that the users tune and improve the sequence design procedure for these particular residues. As an example, the generally constructed layer design protocol may assign some residues to be packed with unrealistic amino acid types within the given context. Also, the blueprint and/or constraint file may need modifications if the original SSE or geometrical requirements do not allow for reasonable side chain packing.

Other backbone and sequence quality control methods

The quality of the protein backbone is crucial for successful protein design. It is suggested that the user always check the quality of the built backbones before and after the sequence design step. Estimating the quality of backbones before sequence design can rely on using Rosetta score terms as mentioned in the above section. As explained above, poor backbone quality distribution could indicate genuine problems in the blueprint inputs, but for some complicated folds to be designed, the problem could be relieved by generating more structures. If increasing the number of generated structures does not help in producing a reasonable amount of backbone structures to proceed with sequence design, the user should consider revisiting the input preparation step.

Some tools capable of checking the backbone quality and its compatibility for the designed sequences are listed here, and the users can find the information at the RosettaCommons website or through the references. The filter '`<worst9mer>`' is a commonly used Rosetta objective to check for the quality of fragments inserted at each residue position before and after the sequence design step. After sequence design, the quality of the backbones should be evaluated in conjunction with the sequence. The deep-learned model, DeepAccNet, is a fast and easy-to-use tool to estimate the quality of the structure for a given sequence (Hiranuma et al., 2020). DeepAccNet also allows the users to check the scores of each residue position. The users can pull out regions with relatively low scores to perform targeted improvement. After the users finish protein design and obtain results with confident scores, it is recommended to use ab initio protein modeling, forward folding, to predict the energy landscape of the protein based on its given structure and sequence (Bradley et al., 2005). This step is usually time consuming and requires more computation capability than the tools listed above and may not be reliable if the protein has complicated topology. This is because there is higher chance the method has limitations in exploring the full conformational space as the degrees of freedom increase. Therefore, it is suggested that the users use forward folding for the last computational checking step, and as a positive indicator to help selecting designs for further wet lab validations. It is not a method to apply in large scale, or after an intermediate design step (i.e. it is not recommended

to forward fold thousands of designs when the designs are not fully optimized, since it would be computationally heavy).

Understanding Results

Please refer to the 'Basic Protocol' and the 'Troubleshooting' sections for results description and analysis.

Time Considerations

The blueprint pipeline can be divided into three steps, 2D map construction, protein backbone construction, and sequence design. The first step, 2D map construction, is the most critical and time-consuming step which will determine the success of the whole protein design. This step requires the user to have a meticulous and/or mathematical understanding of the protein fold class to be designed. While it may be difficult to come up with an accurate blueprint from scratch, the Rosetta-based computational platform makes it possible for the users to exhaust all possible SSE and ABEGO combinations at different locations through constructing many blueprint variants. The backbone and sequence design results based on the blueprints can be used for re-evaluating the original blueprints. As a result, the total computational time will increase depending on the number of different blueprint options to explore or the iterations of designs based on the feedbacks.

The backbone building and sequence design steps for one output usually take 30 to 60 min each on a typical personal computer (4 gigabyte random-access memory, Intel core i5 processor) for the demonstrated β-barrel, which is around 110 amino acids long. Commonly, the larger the protein, the more computations are required for energy calculation, and the longer time it takes to finish each design step. It is suggested that the users use a computer cluster system and run the jobs in parallel if many jobs are to be executed.

The filter steps are usually fast if the scores have been pre-computed and can be completed on a typical personal computer (4 gigabyte random-access memory, Intel core i5 processor) within 10 to 30 min. The specific time consumption is highly dependent on the number of structures and the specific features to be analyzed. Although both the Rosetta score file and the .pdb file format can be used for analysis, when many .pdb files are loaded as poses (i.e. the object that holds the sequence and structural information of a protein in PyRosetta) to be further analyzed using the PyRosetta scripts, the analysis can be delayed due to the time spent on constructing multiple poses. Therefore, if the user is planning to use the scores that have already been evaluated and reported as a part of the backbone or sequence design process, using the score file directly for analysis can save time.

Abbreviations

DSSP - Dictionary of secondary structure of proteins

NTF2 - Nuclear transport factor 2

SSE - secondary structure element

Acknowledgement

This protocol was drafted based on 'De novo design of a fluorescence-activating β-barrel'. Part of the original research was funded by NIH grant R01 GM115545. We thank Prof. David Baker for going through the manuscript and provided suggestions. We acknowledge Dr. Anastassia Vorobieva and Dr. Brian D. Weitzner for their work on developing the concepts and materials for de novo building of β-barrels that were adapted in this protocol. L. A. is supported by 68-2097, the WRF Postdoctoral Sub. G. R. L. is supported by 65-1434 WRF Innovation Fellows, 66-6789 University of Pittsburgh, and 68-0682 Open Phil Imp.

Literature cited

Abkevich, V. I., Gutin, A. M., & Shakhnovich, E. I. (1996). Improved design of stable and fast-folding model proteins. *Fold Des, 1*(3), 221-230. doi:10.1016/s1359-0278(96)00033-8.

Anand, N., Eguchi, R. R., Derry, A., Altman, R. B., & Huang, P.-S. (2020). Protein Sequence Design with a Learned Potential. *bioRxiv*, 2020.01.06.895466. doi:10.1101/2020.01.06.895466.

Anishchenko, I., Chidyausiku, T. M., Ovchinnikov, S., Pellock, S. J., & Baker, D. (2020). De novo protein design by deep network hallucination. *bioRxiv*, 2020.07.22.211482. doi:10.1101/2020.07.22.211482.

Baker, D. (2019). What has de novo protein design taught us about protein folding and biophysics? *Protein Science, 28*(4), 678-683. doi:10.1002/pro.3588.

Basanta, B., Bick, M. J., Bera, A. K., Norn, C., Chow, C. M., Carter, L. P., . . . Baker, D. (2020). An enumerative algorithm for de novo design of proteins with diverse pocket structures. *Proceedings of the National Academy of Sciences, 117*(36), 22135. doi:10.1073/pnas.2005412117.

Bowie, J. U., & Eisenberg, D. (1994). An evolutionary approach to folding small alpha-helical proteins that uses sequence information and an empirical guiding fitness function. *Proceedings of the National Academy of Sciences, 91*(10), 4436-4440. doi:10.1073/pnas.91.10.4436.

Bradley, P., Misura, K. M. S., & Baker, D. (2005). Biochemistry: Toward high-resolution de novo structure prediction for small proteins. *Science, 309*(5742), 1868-1871. doi:10.1126/science.1113801.

Brunette, T. J., Bick, M. J., Hansen, J. M., Chow, C. M., Kollman, J. M., & Baker, D. (2020). Modular repeat protein sculpting using rigid helical junctions. *Proceedings of the National Academy of Sciences, 117*(16), 8870-8875. doi:10.1073/pnas.1908768117.

Cao, L., Goreshnik, I., Coventry, B., Case, J. B., Miller, L., Kozodoy, L., . . . Baker, D. (2020). De novo design of picomolar SARS-CoV-2 miniprotein inhibitors. *Science*, 1-10. doi:10.1126/science.abd9909.

Chaudhury, S., Lyskov, S., & Gray, J. J. (2010). PyRosetta: a script-based interface for implementing molecular modeling algorithms using Rosetta. *Bioinformatics, 26*(5), 689-691. doi:10.1093/bioinformatics/btq007.

Chen, Z., Boyken, S. E., Jia, M., Busch, F., Flores-Solis, D., Bick, M. J., . . . Baker, D. (2019). Programmable design of orthogonal protein heterodimers. *Nature, 565*(7737), 106-111. doi:10.1038/s41586-018-0802-y.

Chevalier, A., Silva, D. A., Rocklin, G. J., Hicks, D. R., Vergara, R., Murapa, P., . . . Baker, D. (2017). Massively parallel de novo protein design for targeted therapeutics. *Nature, 550*(7674), 74-79. doi:10.1038/nature23912.

Dou, J., Vorobieva, A. A., Sheffler, W., Doyle, L. A., Park, H., Bick, M. J., . . . Baker, D. (2018). De novo design of a fluorescence-activating β-barrel. *Nature, 561*(7724), 485-491. doi:10.1038/s41586-018-0509-0.

Fleishman, S. J., Leaver-Fay, A., Corn, J. E., Strauch, E. M., Khare, S. D., Koga, N., . . . Baker, D. (2011). Rosettascripts: A scripting language interface to the Rosetta Macromolecular modeling suite. *PLoS ONE, 6*(6), e20161-e20161. doi:10.1371/journal.pone.0020161.

Fleishman, S. J., Whitehead, T. A., Ekiert, D. C., Dreyfus, C., Corn, J. E., Strauch, E. M., . . . Baker, D. (2011). Computational design of proteins targeting the conserved stem region of influenza hemagglutinin. *Science, 332*(6031), 816-821. doi:10.1126/science.1202617.

Hiranuma, N., Park, H., Anishchanka, I., Baek, M., & Baker, D. (2020). Improved protein structure refinement guided by deep learning based accuracy estimation. *bioRxiv*, 2020.07.17.209643. doi:10.1101/2020.07.17.209643.

Hsia, Y., Mout, R., Sheffler, W., Edman, N. I., Vulovic, I., Park, Y.-J., . . . Baker, D. (2020). Hierarchical design of multi-scale protein complexes by combinatorial assembly of oligomeric helical bundle and repeat protein building blocks. *bioRxiv*, 2020.07.27.221333. doi:10.1101/2020.07.27.221333.

Huang, P. S., Ban, Y. E. A., Richter, F., Andre, I., Vernon, R., Schief, W. R., & Baker, D. (2011). RosettaRemodel: A generalized framework for flexible backbone protein design. *PLoS ONE, 6*(8), e24109-e24109. doi:10.1371/journal.pone.0024109.

Huang, P. S., Feldmeier, K., Parmeggiani, F., Velasco, D. F., Hocker, B., & Baker, D. (2016). De novo design of a four-fold symmetric TIM-barrel protein with atomic-level accuracy. *Nature Chemical Biology, 12*(1), 29-34. doi:10.1038/nchembio.1966.

Jacobs, T. M., Williams, B., Williams, T., Xu, X., Eletsky, A., Federizon, J. F., . . . Kuhlman, B. (2016). Design of structurally distinct proteins using strategies inspired by evolution. *Science, 352*(6286), 687-690. doi:10.1126/science.aad8036.

Kabsch, W., & Sander, C. (1983). Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers, 22*(12), 2577-2637. doi:10.1002/bip.360221211.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., . . . team, J. d. (2016). *Jupyter Notebooks – a publishing format for reproducible computational workflows*. Paper presented at the In Positioning and Power in Academic Publishing: Players, Agents and Agendas, Göttingen, Germany.

Koga, N., Tatsumi-Koga, R., Liu, G., Xiao, R., Acton, T. B., Montelione, G. T., & Baker, D. (2012). Principles for designing ideal protein structures. *Nature, 491*(7423), 222-227. doi:10.1038/nature11600.

Kuhlman, B., Dantas, G., Ireton, G. C., Varani, G., Stoddard, B. L., & Baker, D. (2003). Design of a novel globular protein fold with atomic-level accuracy. *Science, 302*(5649), 1364-1368. doi:10.1126/science.1089427.

Langan, R. A., Boyken, S. E., Ng, A. H., Samson, J. A., Dods, G., Westbrook, A. M., . . . Baker, D. (2019). De novo design of bioactive protein switches. *Nature, 572*(7768), 205-210. doi:10.1038/s41586-019-1432-8.

Lapidoth, G. D., Baran, D., Pszolla, G. M., Norn, C., Alon, A., Tyka, M. D., & Fleishman, S. J. (2015). AbDesign: An algorithm for combinatorial backbone design guided by natural conformations and sequences. *Proteins, 83*(8), 1385-1406. doi:10.1002/prot.24779.

Leaver-Fay, A., O'Meara, M. J., Tyka, M., Jacak, R., Song, Y., Kellogg, E. H., . . . Kuhlman, B. (2013). Scientific benchmarks for guiding macromolecular energy function improvement. *Methods in enzymology, 523*, 109-143. doi:10.1016/B978-0-12-394292-0.00006-0.

Leaver-Fay, A., Tyka, M., Lewis, S. M., Lange, O. F., Thompson, J., Jacak, R., . . . Bradley, P. (2011) Rosetta3: An object-oriented software suite for the simulation and design of macromolecules. In*: Vol. 487. Methods in Enzymology* (pp. 545-574).

Leman, J. K., Weitzner, B. D., Lewis, S. M., Adolf-Bryfogle, J., Alam, N., Alford, R. F., . . . Bonneau, R. (2020). Macromolecular modeling and design in Rosetta: recent methods and frameworks. *Nature Methods, 17*(7), 665-680. doi:10.1038/s41592-020-0848-2.

Lin, Y. R., Koga, N., Tatsumi-Koga, R., Liu, G., Clouser, A. F., Montelione, G. T., . . . DeGrado, W. F. (2015). Control over overall shape and size in de novo designed proteins. *Proceedings of the National Academy of Sciences of the United States of America, 112*(40), e5478-e5485. doi:10.1073/pnas.1509508112.

Marcos, E., Basanta, B., Chidyausiku, T. M., Tang, Y., Oberdorfer, G., Liu, G., . . . Baker, D. (2017). Principles for designing proteins with cavities formed by curved b sheets. *Science, 355*(6321), 201-206. doi:10.1126/science.aah7389.

Marcos, E., Chidyausiku, T. M., McShan, A. C., Evangelidis, T., Nerli, S., Carter, L., . . . Baker, D. (2018). De novo design of a non-local β-sheet protein with high stability and accuracy. *Nature Structural and Molecular Biology, 25*(11), 1028-1034. doi:10.1038/s41594-018-0141-6.

Murzin, A. G., Lesk, A. M., & Chothia, C. (1994a). Principles determining the structure of beta-sheet barrels in proteins. I. A theoretical analysis. *J Mol Biol, 236*(5), 1369-1381. doi:10.1016/0022-2836(94)90064-7.

Murzin, A. G., Lesk, A. M., & Chothia, C. (1994b). Principles determining the structure of beta-sheet barrels in proteins. II. The observed structures. *J Mol Biol, 236*(5), 1382-1400. doi:10.1016/0022-2836(94)90065-5.

Nauli, S., Kuhlman, B., & Baker, D. (2001). Computer-based redesign of a protein folding pathway. *Nature Structural Biology, 8*(7), 602-605. doi:10.1038/89638.

Netzer, R., Listov, D., Lipsh, R., Dym, O., Albeck, S., Knop, O., . . . Fleishman, S. J. (2018). Ultrahigh specificity in a network of computationally designed protein-interaction pairs. *Nature Communications, 9*(1), 5286-5299. doi:10.1038/s41467-018-07722-9.

Pan, X., Thompson, M. C., Zhang, Y., Liu, L., Fraser, J. S., Kelly, M. J. S., & Kortemme, T. (2020). Expanding the space of protein geometries by computational design of de novo fold families. *Science, 369*(6507), 1132-1136. doi:10.1126/science.abc0881.

Pyles, H., Zhang, S., De Yoreo, J. J., & Baker, D. (2019). Controlling protein assembly on inorganic crystals through designed protein interfaces. *Nature, 571*(7764), 251-256. doi:10.1038/s41586-019-1361-6.

Quijano-Rubio, A., Yeh, H.-W., Park, J., Lee, H., Langan, R. A., Boyken, S. E., . . . Baker, D. (2020). De novo design of modular and tunable allosteric biosensors. *bioRxiv*, 2020.07.18.206946. doi:10.1101/2020.07.18.206946.

Schaeffer, R. D., & Daggett, V. (2011). Protein folds and protein folding. *Protein engineering, design & selection, 24*(1-2), 11-19. doi:10.1093/protein/gzq096.

Silva, D. A., Yu, S., Ulge, U. Y., Spangler, J. B., Jude, K. M., Labão-Almeida, C., . . . Baker, D. (2019). De novo design of potent and selective mimics of IL-2 and IL-15. *Nature, 565*(7738), 186-191. doi:10.1038/s41586-018-0830-7.

Simons, K. T., Bonneau, R., Ruczinski, I., & Baker, D. (1999a). Ab initio protein structure prediction of CASP III targets using ROSETTA. *Proteins: Structure, Function and Genetics, 37*(SUPPL. 3), 171-176. doi:10.1002/(SICI)1097-0134(1999)37:3+<171::AID-PROT21>3.0.CO;2-Z.

Simons, K. T., Bonneau, R., Ruczinski, I., & Baker, D. (1999b). Ab initio protein structure prediction of CASP III targets using ROSETTA. *Proteins, Suppl 3*, 171-6. doi:10.1002/(sici)1097-0134(1999)37:3+<171::aid-prot21>3.3.co;2-q.

Simons, K. T., Kooperberg, C., Huang, E., & Baker, D. (1997). Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions. *Journal of Molecular Biology, 268*(1), 209-225. doi:10.1006/jmbi.1997.0959.

Tinberg, C. E., Khare, S. D., Dou, J., Doyle, L., Nelson, J. W., Schena, A., . . . Baker, D. (2013). Computational design of ligand-binding proteins with high affinity and selectivity. *Nature, 501*(7466), 212-216. doi:10.1038/nature12443.

Ueda, G., Antanasijevic, A., Fallas, J. A., Sheffler, W., Copps, J., Ellis, D., . . . Baker, D. (2020). Tailored design of protein nanoparticle scaffolds for multivalent presentation of viral glycoprotein antigens. *eLife, 9*, e57659-e57659. doi:10.7554/eLife.57659.

Wintjens, R. T., Rooman, M. J., & Wodak, S. J. (1996). Automatic classification and analysis of alpha alpha-turn motifs in proteins. *J Mol Biol, 255*(1), 235-253. doi:10.1006/jmbi.1996.0020.

Internet resources

Bash commands: https://www.gnu.org/software/bash/manual/bash.html#Shell-Commands.

All demonstrative files and folders used in this protocol: https://github.com/LAnAlchemist/blueprint_builder_demo

Jupyter Notebook: https://jupyter-notebook.readthedocs.io/en/stable/index.html#

PyRosetta: http://www.pyrosetta.org/

Python 3: https://docs.python.org/3/tutorial/.

Rosetta: https://www.rosettacommons.org/

Figure Legends

Figure 1. An example of a blueprint.

```
1 A L .
0 V EB R
0 V EB R
0 V L R
```

Figure 2. The overall fold, Cβ-strip, and the 2D map of a β-barrel. (a) The cartoon representation of a β-barrel. The backbones are shown in sticks. The carbon, nitrogen, and oxygen atoms are colored in white, blue, and red, respectively. The side chains and hydrogen atoms are omitted for clarity. The backbone hydrogen bonds between each strand are shown in yellow dash lines. (b) A Cβ-strip is shown in sticks and cartoon representation in orange. The carbon, nitrogen, oxygen, and hydrogen atoms are colored in orange, blue, red, and white, respectively. (c) The 2D map of the β-barrel used in this report is shown in cartoon representation. Each circle represents a β-barrel residue. The strand residues are colored in white or gray, and the loop residues are colored in blue. Individual Cβ-strip is distinguished by the color of gray or white. One exemplary Cβ-strip is marked out with a gray dash line. The last strand is also shown in dashed circles to show its complete hydrogen bond registry to the first strand. The N or C terminus of the protein is marked out by 'N' or 'C'. The shear number (S) and strand distance (D) are marked out. Some of the N and C terminal residues in the actual blueprint file used in the report are omitted for clarity. Panels a & b are prepared using PyMOL (The PyMOL Molecular Graphics System, Schrödinger, LLC).

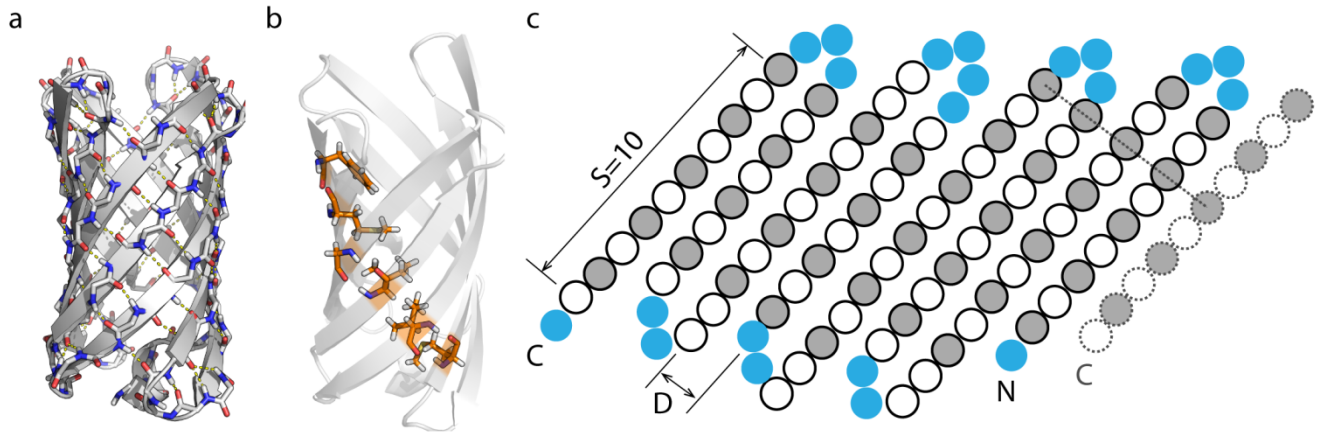

Figure 3. An example of defining a hydrogen bond constraint between the amine of residue 17 and the carbonyl of residue 21.

```
AtomPair N 17 O 21 HARMONIC 3.0 0.5
Angle N 17 H 17 O 21 CIRCULARHARMONIC 3.1 0.3
```