Chapter 3

Site-wise Diversification of Combinatorial Libraries Using Insights from Structure-guided Stability Calculations

Benedikt Dolgikh and Daniel Woldring

Abstract

Many auspicious clinical and industrial accomplishments have improved the human condition by means of protein engineering. Despite these achievements, our incomplete understanding of the sequence–structure–function relationship prevents rapid innovation. To tackle this problem, we must develop and integrate new and existing technologies. To date, directed evolution and rational design have dominated as protein engineering principles. Even so, prior to screening for novel or improved functions, a large collection of variants, within a protein library, exist along an ambiguous mutational terrain. Complicating things further, the choice of where to initialize investigation along a vast sequence space becomes even more difficult given that the majority of any sequence lacks function entirely. Unfortunately, even when considering functionally relevant positions, random substitutions can prove to be destabilizing, causing a hindrance to an otherwise function-inducing, stability-reliant folding process. To enhance productivity in the field, we seek to address this issue of destabilization, and subsequent disfunction, at protein–protein and protein–ligand interacting regions. Herein, the process of choosing amenable positions – and amino acids at those positions – allows for a refined, knowledge-based approach to combinatorial library design. Using structural data, we perform computational stability prediction with FoldX's PositionScan and Rosetta's ddG_monomer in tandem, allowing for the refinement of our thermodynamic stability data through the comparison of results. In turn, we provide a process for selecting in silico predicted mutually stabilizing positions and avoiding overly destabilizing ones that guides the site-wise diversification of combinatorial libraries.

Key words Computational, Protein engineering, Stability, Library design, Site-wise diversification

1 Introduction

Engineering novel or improved protein function often requires making several mutations at various positions. Given that random mutations tend to be destabilizing [1] and that most natural proteins are only marginally stable to begin with, designing functional protein variants necessitates the exclusion of overly destabilizing mutations. Identifying the most mutationally amenable positions is assisted by computational modeling tools such as Rosetta's

Michael W. Traxlmayr (ed.), *Yeast Surface Display*, Methods in Molecular Biology,
vol. 2491, https://doi.org/10.1007/978-1-0716-2285-8_3,

ddg_monomer [2] and ddg_cartesian [3] applications or FoldX PositionScan and BuildModel commands [4] (*see* **Note 1**). To screen proteins with computational tools for engineered design and functionality, a computational approach is limited by the type of data, that is a structural model or protein sequence, and the number or caliber of extractable properties, such as high-resolution structural data or sufficiently large deep sequencing datasets. In locating amenable candidates for library design, a stability prediction tool greatly depends on the quality of input structure [5]. Fortunately, the physicochemical terms that influence tertiary structure are present in high-resolution structural information and necessary for energy function calculations [6, 7]. By modeling the mechanics of functional regions to explore their potential for thermodynamic variation, structurally guided stability prediction detects positions that are most and least amenable for site-wise diversification of combinatorial libraries [8].

Stability prediction tools iteratively sample an ensemble of conformations from high-resolution structural information, applying an energy function or force field to acquire the most stable fold of a given protein [2, 9]. Energy functions for stability prediction can be varied in their accuracy (*see* **Note 2**) and computational resource usage [10, 11] (*see* **Note 3**) in a multitude of use cases [12–18]. In our case, computationally predicted stability difference ($\Delta\Delta G$) between wildtype and mutant folds helps to understand the mode for how variation in stability influences protein folding and unfolding. Going one step further in our methodology, a recent comparison of experimental stability data to the combined prediction of FoldX and Rosetta has rationalized the efficacy of combinatorial approaches [19]. Recently, the discussion of integrating multiple computational tools for protein engineering has gained momentum [20–22]. By maximizing the utility between two stability prediction tools, we present the opportunity to seek out additional computational approaches (Chapters 4 and 5 of this volume); the drawbacks of separate datatypes still converge at their source system, biomolecule, or protein.

To achieve more accurate predictions we combine the data from FoldX, a user friendly tool [23] for researchers with little computational experience, and Rosetta's ddg_monomer, implementing backbone flexibility in structural minimization that parallels more realistic structural changes upon mutagenesis [2]. Overall, the informative, albeit imperfect [12, 19, 24, 25], stability prediction capabilities of Rosetta and FoldX force fields enable site-wise diversification, creating a platform for combinatorial libraries [26–29]. Here, we locate positions of interest around a known active site location, followed by our general protocols for performing stability prediction within Rosetta and FoldX (Fig. 1).

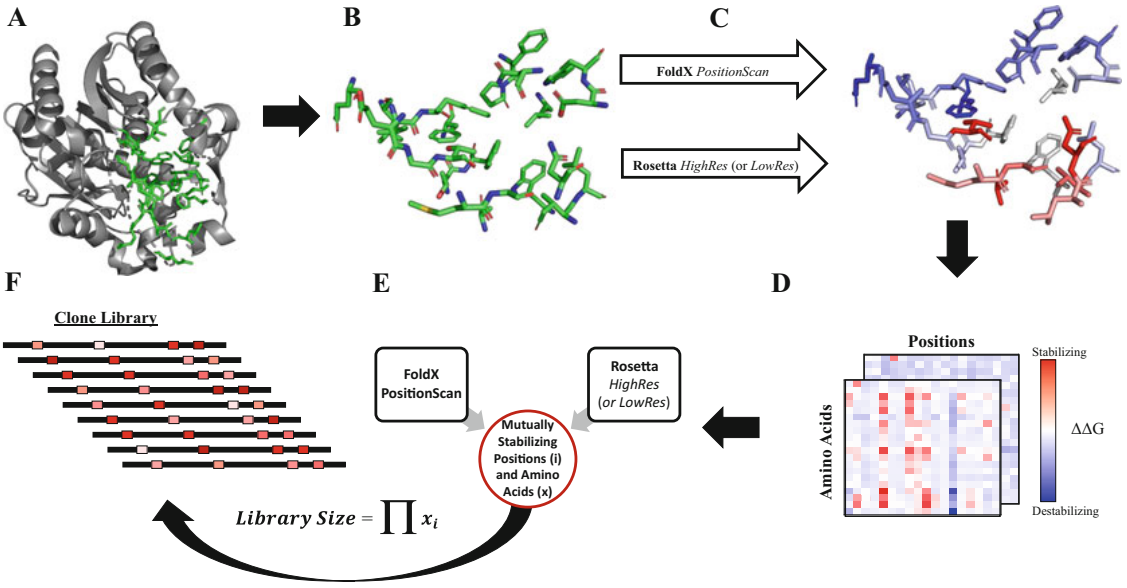

Fig. 1 Computational stability prediction workflow. (**a**) The active site (Green) within an enzyme is evaluated for mutational tolerance. (**b**) Residues proximal to the active site are chosen. (**c**) Multiple stability prediction tools independently calculate all possible point mutations surrounding the active site resulting in a range of stabilizing (Red) and destabilizing (Blue) effects. (**d**) Graphical analysis of ΔΔG data provides insights for rational library design. (**e**) The two datasets inform library size by the product of the number of mutually stabilizing amino acids (*x*) offered at each position of interest (*i*). (**f**) Library construction with overlap extension PCR uses degenerate codon oligonucleotides to introduce diversity. Protein and residue images were created using PyMOL

2 Materials

1. A UNIX-based operating system available on your local desktop or through a computing cluster (*see* **Note 2**) for running command line and python scripts (*see* **Note 4**).
2. Download and unpack FoldX 5.0 (http://foldxsuite.crg.eu/).
3. Download, unpack, and compile Rosetta 3.12 source + binaries (https://www.rosettacommons.org/) (*see* **Notes 4** and **5**).
4. PDB structure file(s) for a protein of interest (*see* **Note 6**).
5. Download and unpack PyMOL (https://pymol.org/2/).

3 Methods

3.1 Choosing Residues of Interest

1. Open PyMOL and load PDB structures for your protein of interest.
2. Locate the active site, then highlight positions within 5 Å of the selection using:

```
show sticks, byres all within 5 of <SELECTION_NAME>
```

(*see* **Notes 7** and **8**).

3. Record and use these positions of interest in all downstream stability prediction.

3.2 FoldX Stability Prediction

1. Using a text editor (*see* **Note 9**), create a file called

```
RepairPDB_<pdbname>.cfg.
```

2. Use *.cfg file templates* from GitHub to optimize your configuration file (*see* **Note 10**).
3. Run the FoldX RepairPDB command (*see* **Note 11**) for each PDB structure separately using:

```
./foldx -f ./RepairPDB_<pdbname>.cfg
```

4. Repeat **steps 1** and **2** but name the .cfg file `PS_<pdbname>_Repair.cfg`.
5. Run the FoldX PositionScan command (*see* **Note 12**) for each PDB structure separately using:

```
./foldx -f ./PS_<pdbname>_Repair.cfg
```

6. Use the $\Delta\Delta G_{unfolding}$ (kcal/mol) values from `PS_<pdbname>_Repair_scanning_output.txt` (second column) for downstream graphical analyses and degenerate codon design (Subheading 3.4). Examples are provided on GitHub (*see* **Note 10**).

3.3 Rosetta ddg_monomer Stability Predicition

3.3.1 Structural File Preparation for High- and Low-Resolution ddg_monomer

1. Clean your PDB structure file:

```
python2.7 \
          /PATH/TO/ROSETTA/tools/protein_tools/clean_pdb.py \
           ./<pdbname>.pdb ChainID
```

(*see* **Note 13**)

2. Using the `<pdbname>_<chainid>.pdb` output file, renumber your PDB structure:

```
python2.7 \
                /PATH/TO/ROSETTA/tools/protein_tools/scripts/
pdb_renumber.py  \  ./<pdbname>_<chainid>.pdb
<renumberedpdbname>.pdb
```

3. Using a text editor, examine the position number (column 6) in `<renumberedpdbname>.pdb` and make note of the position number for your positions of interest if they have changed (*see* **Note 10**).
4. Use *.resfile templates* from GitHub to optimize your mutfile/resfile (*see* **Note 10**).

3.3.2 Low-Resolution ddg_monomer: (See **Note 14**)

1. Run the ddg_monomer.linuxgccrelease (*see* **Note 15**) executable (*see* **Note 16**) in the directory with all the necessary files:

```
/PATH/TO/ROSETTA/main/source/bin/ddg_monomer.linuxgccrelease
\
            -in:file:s ./<renumberedpdbname>.pdb \
            -ddg::weight_file soft_rep_design \
            -ddg::iterations 50 \
            -ddg::dump_pdbs true \
            -database /PATH/TO/ROSETTA/main/database/ \
            -ddg::local_opt_only true \
            -ddg::min_cst false \
            -ddg::mean true \
            -ddg::min false \
            -ddg::sc_min_only true \
            -in:file:fullatom \
            -resfile ./<resfilename>.resfile \
            -ignore_zero_occupancy false \
            -ignore_unrecognized_res
```

2. Use the $\Delta\Delta G_{folding}$ (REU) (*see* **Note 17**) values from the `ddg_predictions.out` (first value in row; "total") file for downstream graphical analyses and degenerate codon design (Subheading 3.4). Examples are provided on GitHub (*see* **Note 10**).

3.3.3 High-Resolution ddG_monomer

1. Create a list file for your cleaned and renumbered pdb structure:

```
ls ./<renumberedpdbname>.pdb > <listname>
```

2. Run the `minimize_with_cst.linuxgccrelease` executable (*see* **Note 16**) in the directory with all necessary files:

```
/PATH/TO/ROSETTA/main/source/bin/minimize_with_cst.linuxgc-
crelease \
            -in:file:l ./<listname> \
            -in:file:fullatom \
            -ignore_unrecognized_res \
            -fa_max_dis 9.0 \
```

```
            -database /PATH/TO/ROSETTA/main/database/ \
            -ddg::harmonic_ca_tether 0.5 \
            -score:weights pre_talaris_2013_standard \
            -restore_pre_talaris_2013_behavior \
            -ddg::constraint_weight 1.0 \
            -ddg::out_pdb_prefix min_cst_0.5 \
            -ddg::sc_min_only false \
                 -score:patch /PATH/TO/ROSETTA/main/database/
scoring/weights/score12.wts_patch > mincst.log
```

3. After pre-minimization, convert the `mincst.log` file to a distance-restraint file (*see* **Notes 10** and **18**):

```
tcsh ./convert_to_cst_file.sh ./mincst.log >
<constraintfilename>.cst
```

4. Run the `ddg_monomer.linuxgccrelease` executable in the directory with all necessary files:

```
/PATH/TO/ROSETTA/main/source/bin/ddg_monomer.linuxgccrelease
\
            -in:file:s ./min_cst_0.5.<renumberedpdbname>_001.
pdb \
            -ddg::weight_file soft_rep_design \
           -ddg:minimization_scorefunction pre_talaris_2013_-
standard \
            -restore_pre_talaris_2013_behavior \
              -ddg::minimization_patch /PATH/TO/ROSETTA/main/
database/scoring/weights/score12.wts_patch \
            -ddg::iterations 50 \
            -ddg::dump_pdbs true \
            -database /PATH/TO/ROSETTA/main/database/ \
            -ignore_unrecognized_res \
            -ddg::local_opt_only false \
            -ddg::min_cst true \
            -ddg::mean false \
            -ddg::min true \
            -ddg::sc_min_only false \
            -in:file:fullatom \
            -resfile ./<renumberedpdbname>.resfile \
            -ignore_zero_occupancy false \
            -ddg::ramp_repulsive true \
          -constraints::cst_file ./<constraintfilename>.cst \
            -ddg:suppress_checkpointing true
```

5. Use the $\Delta\Delta G_{folding}$ (REU) values from the `ddg_predictions.out` (first value in row, "total") file for downstream

graphical analyses and degenerate codon design (Subheading 3.4). Examples are provided on GitHub (*see* **Note 10**).

3.4 Library Design for Overlap Extension PCR and Electroporation into Yeast

1. For library design, at each analyzed position, enumerate the number of residues that balance the stringency of stabilizing mutations (e.g. between $\Delta\Delta G < -1.5$–0 kcal/mol) with the estimated library size (e.g. 10^9 for Yeast Surface Display) to cater to the library design goals. Create a list of individual positions and the corresponding amino acids that are most amenable to mutation (i.e., sites that have the greatest number of stabilizing mutations). Aggregate these data with the results from both stability prediction tools.
2. Based on the created list, identify a degenerate codon that encodes for an identical or similar distribution of amino acids for each site (*see* **Note 19**). Examples are provided on GitHub (*see* **Note 10**).
3. Order oligonucleotide sequences that incorporate degenerate codons at the position(s) determined in **step 1**.
4. Construct full length gene using overlap extension PCR [27, 30–32]. Electroporate the newly constructed insert library with an appropriate, linearized yeast surface display vector (e.g., pCTcon2 [33]) [26–29, 34].

4 Notes

1. Web-servers such as FireProt [35], PoPMuSiC [36], ELASPIC [37], DUET [38], and HotSpotWizard [39] reduce computational demand from a user's home desktop. There also exist combinatorial tools such as FRESCO [40], which has been optimized for integration of FoldX and Rosetta stability prediction in combination with the Dynamic Disulfide Discovery algorithm.
2. Limitations of accuracy: To further elaborate on this considerable limitation in stability prediction tools it is important to understand the breadth of variable training data. Energy functions have been trained using experimental data where many variants contain alanine substitution mutations, such as in FoldX, and many training datasets largely consist of destabilizing mutations.
3. Computing cluster resources are highly recommended for performing these commands. Rosetta's ddg_monomer utilizes significant resources compared to FoldX, but both tools increase computational demand with the number of mutated positions, the mutational approach (e.g., all 20 amino acids, only charged amino acids, etc.), and iterations of structural energy minimization and optimization.

4. Version information for other necessary modules: Python – 2.7.16 – https://www.python.org/; Biopython – 1.75 (Use `pip install`. For pdb_renumber.py); GCC – 8.3.0 - https://gcc.gnu.org/; OpenMPI – 3.1.4 - https://www.open-mpi.org/
5. Rosetta is parallelizable; calculation speed mainly depends on a Rosetta application's utilization of computational resources (https://www.rosettacommons.org/docs/latest/build_documentation/Build-Documentation).
6. The number of structures analyzed is dependent on structures available through databases such as https://www.rcsb.org/, but should ideally be as many as are available.
7. To make a selection, simply click on a residue(s) and change the selection name by moving the mouse over to *(sele)*, click *A* for Action, click *rename selection*, rename, and click ENTER on your keyboard (refer to **Note 10**). After running the command, consider starting with 20 positions to limit computational demand; adjust as needed.
8. Any code is in this `FONT`. Any code in these `<* >` brackets should be replaced with the name of your choice (example: `<renumberedpdbname>.pdb` = `1A2B.pdb`). The. / before a command indicates that the file or executable is in the current directory.
9. A frequently used text editor in UNIX is called *vim*. It will allow you to read or create .txt, .cfg, .pdb, .out, .log, .resfile, and other files containing text. To use vim, type:

```
vi <filename>.txt (example:  vi <renumberedpdbname>.pdb)
```

   To edit a vim file, press `A`. Exit editing by `ESC`. Save and exit your file by typing: `wq`.
10. Templates and examples for input and output files/data can be found here: https://GitHub.com/WoldringLabMSU/Computational-Stability
11. More information about RepairPDB can be found here: http://foldxsuite.crg.eu/command/RepairPDB
12. More information about PositionScan can be found here: http://foldxsuite.crg.eu/command/PositionScan
13. Long commands are separated using a \ back slash to indicate that the line continues.
14. The authors of ddg_monomer indicate many approaches for stability prediction with their tool [2], two of which are used here. The major difference between both is the addition of distance-restraints for backbone flexibility during high-

resolution minimization. As a result, the high-resolution method requires considerable computational time compared to the low-resolution method. Regardless of approach, using FoldX with either of them is still useful for improving predictive accuracy [19].

15. Rosetta executable nomenclature varies depending on the Rosetta package that you downloaded and the coinciding method for compiling. While `ddg_monomer.linuxgccrelease` is most referenced, variations exist such as `ddg_monomer.static.linuxgccrelease` or `ddg_monomer.default.linuxgccrelease`. For more information refer to **Note 9**.
16. More information about ddg_monomer can be found here: https://www.rosettacommons.org/docs/latest/application_documentation/analysis/ddg-monomer
17. Rosetta uses arbitrary units that replace kcal/mol. The energy function and corresponding units we have chosen to use for our protocol is optimized for this Rosetta application. Newer energy functions and those that correlate more closely to kcal/mol can be explored in recent literature on the Rosetta scorefunction [41].
18. The tcsh command may not work on a computing cluster without administrator permissions, so you will need to transfer the `convert_to_cst_file.sh` file and the `mincst.log` file to your local desktop. To copy files from a server to a local desktop or otherwise, the `scp -r` command arguments will vary depending on your user ID (refer to *Navigating Rosetta to run executables* from **Note 10**).
19. When designing degenerate codons for a particular position, while the amino acid distribution encoded by the degenerate bases may not perfectly match the intended distribution of residues predicted to be most stabilizing at that site, care should be taken to avoid including residues that were predicted to be overly destabilizing (e.g. $\Delta\Delta G > 1.5$ kcal/mol) [27]. Online tools such as SwiftLib are available to guide the degenerate codon design process [42].

References

1. Baase WA, Liu L, Tronrud DE et al (2010) Lessons from the lysozyme of phage T4. Protein Sci 19:631–641
2. Kellogg EH, Leaver-Fay A, Baker D (2011) Role of conformational sampling in computing mutation-induced changes in protein structure and stability. Proteins 79:830–838
3. Park H, Bradley P, Greisen P et al (2016) Simultaneous optimization of biomolecular energy functions on features from small molecules and macromolecules. J Chem Theory Comput 12:6201–6212
4. Delgado J, Radusky LG, Cianferoni D et al (2019) FoldX 5.0: working with RNA, small molecules and a new graphical interface. Bioinformatics 35:4168–4169
5. Davey JA, Chica RA (2015) Optimization of rotamers prior to template minimization improves stability predictions made by

computational protein design. Protein Sci 24: 545–560

6. Buß O, Rudat J, Ochsenreither K (2018) FoldX as protein engineering tool: better than random based approaches? Comput Struct Biotechnol J 16:25–33
7. Hou T, Wang J, Li Y et al (2011) Assessing the performance of the MM/PBSA and MM/GBSA methods. 1. The accuracy of binding free energy calculations based on molecular dynamics simulations. J Chem Inf Model 51: 69–82
8. Tokuriki N, Stricher F, Serrano L et al (2008) How protein stability and new functions trade off. PLoS Comput Biol 4:e1000002
9. Naganathan AN (2019) Modulation of allosteric coupling by mutations: from protein dynamics and packing to altered native ensembles and function. Curr Opin Struct Biol 54: 1–9
10. Pohorille A, Jarzynski C, Chipot C (2010) Good practices in free-energy calculations. J Phys Chem B 114:10235–10253
11. Klimovich PV, Shirts MR, Mobley DL (2015) Guidelines for the analysis of free energy calculations. J Comput Aided Mol Des 29:397–411
12. Strokach A, Corbi-Verge C, Kim PM (2019) Predicting changes in protein stability caused by mutation using sequence-and structure-based methods in a CAGI5 blind challenge. Hum Mutat 40:1414–1423
13. Mazurenko S (2020) Predicting protein stability and solubility changes upon mutations: data perspective. ChemCatChem 12:5590–5598
14. Beauchamp KA, Lin YS, Das R et al (2012) Are protein force fields getting better? A systematic benchmark on 524 diverse NMR measurements. J Chem Theory Comput 8:1409–1414
15. Pucci F, Bernaerts KV, Kwasigroch JM et al (2018) Quantification of biases in predictions of protein stability changes upon mutations. Bioinformatics 34:3659–3665
16. Thiltgen G, Goldstein RA (2012) Assessing predictors of changes in protein stability upon mutation using self-consistency. PLoS One 7: 46084
17. Huang P, Chu SKS, Frizzo HN et al (2020) Evaluating protein engineering thermostability prediction tools using an independently generated dataset. ACS Omega 5:6487–6493
18. Kumar V, Rahman S, Choudhry H et al (2017) Computing disease-linked SOD1 mutations: deciphering protein stability and patient-phenotype relations article. Sci Rep 7:1–13
19. Nisthal A, Wang CY, Ary ML et al (2019) Protein stability engineering insights revealed by domain-wide comprehensive mutagenesis. Proc Natl Acad Sci U S A 116:16367–16377
20. Adolf-Bryfogle J, Teets FD, Bahl CD (2021) Toward complete rational control over protein structure and function through computational design. Curr Opin Struct Biol 66:170–177
21. Sun J, Cui Y, Wu B (2021) GRAPE, a greedy accumulated strategy for computational protein engineering. In: Methods in enzymology. Academic, pp 207–230
22. Soni S (2021) Trends in lipase engineering for enhanced biocatalysis. Biotechnol Appl Biochem 59:13204–13231
23. Van DJ, Delgado J, Stricher F et al (2011) A graphical interface for the FoldX forcefield. Bioinformatics 27:1711–1712
24. Khan S, Vihinen M (2010) Performance of protein stability predictors. Hum Mutat 31: 675–684
25. Potapov V, Cohen M, Schreiber G (2009) Assessing computational methods for predicting protein stability upon mutation: good on average but not in the details. Protein Eng Des Sel 22:553–560
26. Woldring DR, Holec PV, Zhou H et al (2015) High-throughput ligand discovery reveals a sitewise gradient of diversity in broadly evolved hydrophilic fibronectin domains. PLoS One 10:e0138956
27. Woldring DR, Holec PV, Stern LA et al (2017) A gradient of sitewise diversity promotes evolutionary fitness for binder discovery in a three-helix bundle protein scaffold. Biochemistry 56: 1656–1671
28. Kruziki MA, Bhatnagar S, Woldring DR et al (2015) A 45-amino-acid scaffold mined from the PDB for high-affinity ligand engineering. Chem Biol 22:946–956
29. Kruziki MA, Sarma V, Hackel BJ (2018) Constrained combinatorial libraries of Gp2 proteins enhance discovery of PD-L1 binders. ACS Comb Sci 20:423–435
30. Bryksin AV, Matsumura I (2010) Overlap extension PCR cloning: a simple and reliable way to create recombinant plasmids. BioTechniques 48:463–465
31. Schimming O, Fleischhacker F, Nollmann FI et al (2014) Yeast homologous recombination cloning leading to the novel peptides ambactin and xenolindicin. Chembiochem 15: 1290–1294
32. An Y, Ji J, Wu W et al (2005) A rapid and efficient method for multiple-site mutagenesis with a modified overlap extension PCR. Appl Microbiol Biotechnol 68:774–778

33. Chao G, Lau WL, Hackel BJ et al (2006) Isolating and engineering human antibodies using yeast surface display. Nat Protoc 1:755–768
34. Benatuil L, Perez JM, Belk J et al (2010) An improved yeast transformation method for the generation of very large human antibody libraries. Protein Eng Des Sel 23:155–159
35. Bednar D, Beerens K, Sebestova E et al (2015) FireProt: energy- and evolution-based computational design of thermostable multiple-point mutants. PLoS Comput Biol 11:e1004556
36. Dehouck Y, Kwasigroch JM, Gilis D et al (2011) PoPMuSiC 2.1: a web server for the estimation of protein stability changes upon mutation and sequence optimality. BMC Bioinformatics 12:151
37. Witvliet DK, Strokach A, Giraldo-Forero AF et al (2016) ELASPIC web-server: proteome-wide structure-based prediction of mutation effects on protein stability and binding affinity. Bioinformatics 32:1589–1591
38. Pires DEV, Ascher DB, Blundell TL (2014) DUET: a server for predicting effects of mutations on protein stability using an integrated computational approach. Nucleic Acids Res 42:W314
39. Sumbalova L, Stourac J, Martinek T et al (2018) HotSpot Wizard 3.0: web server for automated design of mutations and smart libraries based on sequence input information. Nucleic Acids Res 46:W356–W362
40. Wijma HJ, Fürst MJLJ, Janssen DB (2018) A computational library design protocol for rapid improvement of protein stability: FRESCO. In: Methods in molecular biology. Humana Press, pp 69–85
41. Alford RF, Leaver-Fay A, Jeliazkov JR et al (2017) The rosetta all-atom energy function for macromolecular modeling and design. J Chem Theory Comput 13:3031–3048
42. Jacobs TM, Yumerefendi H, Kuhlman B et al (2015) SwiftLib: rapid degenerate-codon-library optimization through dynamic programming. Nucleic Acids Res 43:e34