

OSPREY

User Manual

v 1.0

Ivelin Georgiev and Bruce R. Donald

Copyright (C) 2001-2009 Bruce Donald Lab, Duke University

Contents

1	Introduction	2
1.1	Modeling Flexibility	3
2	Installation	6
3	Setting up OSPREY	8
3.1	Compute Node Setup	8
3.2	Starting OSPREY	9
3.2.1	Main Configuration File	10
4	OSPREY Input Model	13
4.1	Setting up the Input Structure	13
4.2	Rotamer Libraries	15
4.3	Energy Function	17
5	OSPREY Commands	22
5.1	GMEC-based Redesign	24
5.1.1	Configuration Files	24
5.1.2	Output Files	32
5.1.3	Performing Backrubs	36
5.2	Redesign Using K^*	38
5.2.1	Configuration Files	39
5.2.2	Output Files	45
5.3	Residue Entropy Computation	49
5.3.1	Configuration Files	49
5.3.2	Output Files	50
6	Special Types of Redesign	53
6.1	Protein-Protein and Protein-Peptide Redesign	53
6.2	Modeling Explicit Waters	54
7	OSPREY via an Example	56
A	OSPREY Class Summary	71

Chapter 1

Introduction

OSPREY (Open Source Protein REdesign for You) is a suite of programs for computational structure-based protein design. OSPREY is developed in the lab of Prof. Bruce Donald at Duke University. This user manual is for version **1.0** of the software.

OSPREY is free software and can be redistributed and/or modified under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (optionally) any later version. OSPREY is distributed in the hope that it will be useful, but **WITHOUT ANY WARRANTY**; without even the implied warranty of **MERCHANTABILITY** or **FITNESS FOR A PARTICULAR PURPOSE**. See the GNU Lesser General Public License for more details. Full licensing details, including citation requirements for the various different modules of the software, are found in the document **license.pdf** enclosed with this package distribution.

OSPREY is specifically designed to identify protein mutants that possess desired target properties (e.g., improved stability, switch of substrate specificity, etc.). OSPREY can also be used for predicting small-molecule inhibitors. With certain limitations, the current version of OSPREY can also model protein-protein and protein-peptide interactions. OSPREY is built around the following algorithmic modules:

- **DEE/A***: provably-accurate algorithms for protein design that combine Dead-End Elimination (DEE) rotamer pruning [6, 18] with A^* conformation enumeration [14]. The DEE/ A^* algorithms score and rank mutation sequences based on the single best conformation for each sequence, the Global Minimum Energy Conformation (GMEC). Hence, these algorithms are referred to as *GMEC-based*. The DEE/ A^* algorithms are typically applied to redesign specific parts of the protein (e.g., the protein core). For problems where the goal is to improve protein-ligand interactions, the K^* algorithm is typically used instead (see below). **NOTE:** In this documentation, the term *DEE* refers to all of *traditional DEE* [6, 18] (DEE for rigid rotamers and a rigid backbone), *MinDEE* [10] (minimized DEE for continuously-flexible rotamers and a rigid backbone), *BD* [7] (DEE for continuously-flexible backbones), and *BRDEE* [8] (DEE for backrub protein motions). In cases where a specific DEE algorithm is referenced, the corresponding algorithm name (e.g., *BRDEE*) is used explicitly.
- **K^*** : a provably-accurate algorithm for protein-ligand binding prediction and enzyme re-design [1, 10, 15]. K^* computes a provably-accurate approximation (given the input model, see below) to the binding constant for a given protein-ligand complex by computing partition functions over ensembles of (energy-minimized) conformations for the bound protein-

ligand complex and the unbound protein and ligand. Hence, the K^* algorithm is referred to as *ensemble-based*. K^* is typically applied to predict mutations to protein binding/active residues in order to switch the substrate specificity toward a novel substrate [1]. K^* can also be applied to design small-molecule inhibitors for a given protein (or set of proteins).

- **SCMF**: a Self-Consistent Mean Field (SCMF) algorithm for computing the entropy of each residue position in a protein [1, 19]. This algorithm can be used as part of a hybrid mutation search for enzyme redesign that also incorporates K^* and DEE/ A^* : K^* can be applied first to predict mutations to the enzyme active site that improve the target substrate specificity; SCMF can then be used to identify mutable positions anywhere in a protein, both close to and far from the active site of an enzyme; finally, DEE/ A^* can be applied to predict mutations to these mutable positions for further improvement in the target substrate specificity [1].

The basic data and algorithm flow in OSPREY is summarized in Fig. 1.1. The input model for the OSPREY modules consists of an input structure for redesign, rotamer libraries for proteins and general compounds (e.g., small-molecule inhibitors), and a pairwise energy function for scoring and ranking the computational predictions. Additionally, input configuration files specify required mutation search parameters for the different modules. Computed structures for selected K^* and DEE/ A^* mutant predictions can also be generated for further visual and structural analysis by the user. OSPREY uses MPI for distributed computation.

1.1 Modeling Flexibility

OSPREY is capable of modeling additional protein and ligand flexibility as compared to other structure-based design approaches. Typically, protein design algorithms use a model with a rigid protein backbone and rigid rotamers [6, 18]. In contrast, OSPREY is capable of modeling continuous side-chain flexibility (i.e., flexible rotamers) [10] and continuous [7] or discrete [8] backbone flexibility. The user can select to model different types of flexibility by appropriately manipulating some configuration file parameters (see Sec. 5.1.1). Additionally, flexibility in OSPREY can be modeled using conformational ensembles, as in the K^* algorithm [1, 10]. The K^* module is described in Sec. 5.2.

Organization

This documentation is organized as follows:

- Chapter 2 contains installation instructions for OSPREY and other required software packages.
- Instructions for initializing and starting up OSPREY are provided in Chapter 3.
- The three parts of the OSPREY input model (input structure, rotamer libraries, and energy function) are described in Chapter 4.
- The various algorithmic modules (DEE/ A^* , K^* , and SCMF) are described in Chapter 5, along with the corresponding input configuration files and instructions for generating structures for selected mutants predicted by the algorithms.

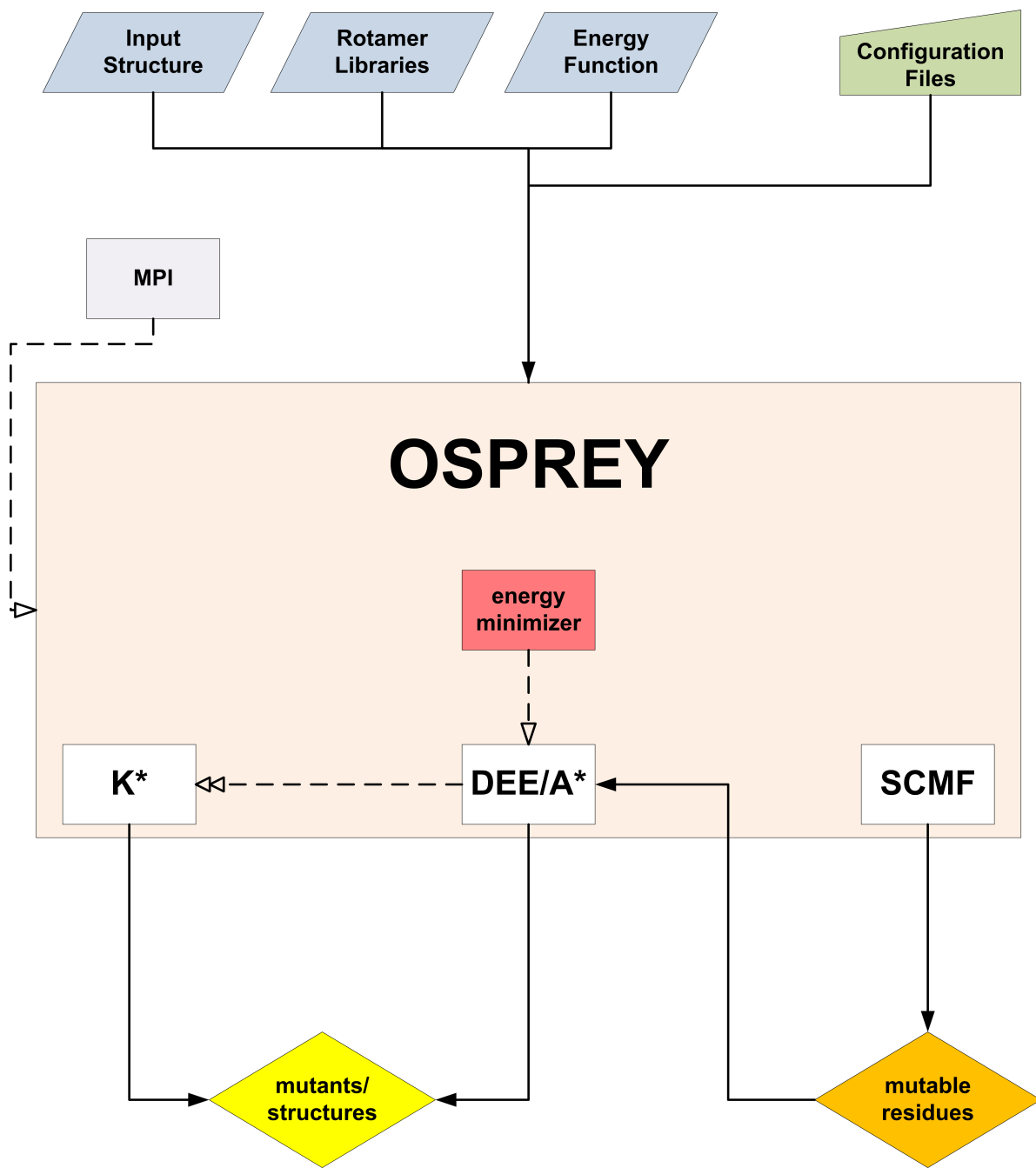


Figure 1.1: Basic data and algorithm flow in OSPREY.

- Chapter 6 provides instructions for applying OSPREY to two special cases of redesign problems: modeling protein-protein and protein-peptide interactions and modeling explicit water molecules.
- A detailed walk-through of an actual protein redesign example using K^* , from input setup to analysis of the results, is described in Chapter 7.
- Appendix A presents a brief overview of the OSPREY classes.

The primary contributors to this version of the OSPREY distribution are: Ivelin Georgiev, Ryan Lilien, and Bruce Donald.

Chapter 2

Installation

To be able to run OSPREY, the following additional programs must also be installed: Java, MPICH2 (<http://www-unix.mcs.anl.gov/mpi/mpich2/index.htm>), and mpiJava (<http://www.hpjava.org/mpiJava.html>). Following the default installation instructions for these programs should be sufficient on 32-bit machines and some 64-bit machines. On certain 64-bit machines, however, the following modified installation instructions must be used. These instructions assume that Java, MPICH2, and mpiJava will be installed as subdirectories in '/home/you/mpi/' (modify this path according to your preference).

(1) **Installing 64-bit Java (v. jdk 1.6.0_06).** Follow the default installation instructions. Update your path to make sure that this java version comes first in your path:

```
export PATH=/home/you/mpi/jdk1.6.0_06/bin:$PATH
```

(2) **Modifying Java.** For the mpiJava installation (see below), you may need to copy the file 'jni_md.h' from 'jdk1.6.0_06/include/linux/' to 'jdk1.6.0_06/include/':

```
cd /home/you/mpi/jdk1.6.0_06/include/linux
cp -i jni_md.h ../
```

(3) **Installing MPICH2 (v. 1.0.7).** Using bash:

```
tar -xzf mpich2-1.0.7.tar.gz
cd mpich2-1.0.7
export CFLAGS="-fPIC"
./configure --prefix=/home/you/mpi/mpich2-install --enable-sharedlibs=gcc
make
make install
export PATH=/home/you/mpi/mpich2-install/bin:$PATH
```

(4) **Installing mpiJava (v. 1.2.5).** Using bash:

```
tar -xzf mpiJava-1.2.5.tar.gz
cd mpiJava
```

```
export DEFPINS="-shared -fPIC"
export LDFLAGSIG="-shared -fPIC"
export LDFLAG="-shared -fPIC"
./configure --with-MPI=mpich
make
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/you/mpi/mpiJava/lib
export CLASSPATH=$CLASSPATH:/home/you/mpi/mpiJava/lib/classes
```

NOTE: `./configure --with-MPI=mpich` for mpiJava may generate some errors/warnings: these can be generally discarded; if at the end of 'make' there are no errors, then the mpiJava installation should be successful.

(5) Installing OSPREY. After Java, mpich2, and mpiJava have been successfully installed, choose a directory where the OSPREY software will be installed, copy all files to that directory, and compile using:

```
javac *.java
```

Some warning statements may be output when `javac` is called, but these can be generally discarded. OSPREY should now be installed and ready for use.

Chapter 3

Setting up OSPREY

3.1 Compute Node Setup

After the installation of all required software is complete (see Chapter 2), OSPREY will be ready for use. First, MPI must be setup to run on a selected set of compute nodes. Users who are familiar with MPI may skip to Sec. 3.2. Next, some basic MPI functionality that should be sufficient for the proper execution of OSPREY is described.

We will assume that the `.mpd.conf` file has been created and saved according to the instructions in the MPICH2 Installer's Guide. We will also assume that the list of available compute nodes is stored in the file `mpd.hosts` in the OSPREY code directory. Each line in the `mpd.hosts` file corresponds to a single compute node. An example `mpd.hosts` file may look like this:

```
linux1
linux2
linux3
linux4
linux5
```

In this example, there are five compute nodes on which MPI will be started. The user must make sure that they can `ssh` into any of these nodes without having to enter a password. One possible way to do this is to first execute the following commands and then manually login to each of the selected nodes:

```
ssh-keygen -t rsa
cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
```

The following command will set up MPI for the list of nodes in `mpd.hosts`:

```
mpdboot -n 5 -f mpd.hosts
```

The number `5` for the `-n` argument is the total number of compute nodes on which MPI should be started; in this case, this number is equal to the total number of nodes in the `mpd.hosts` file. If `mpdboot` is executed from a node not in the `mpd.hosts` file (e.g., `linux6`), then `-n` could be called with a value of `6` (or less, in which case MPI will be started on only a subset of the nodes in the `mpd.hosts` file). To check whether MPI was successfully started on the given set of nodes, the user

can execute the following command:

mpdtrace

This command should output the names of all of the nodes on which MPI should have been started. At this point, the user has created a ring of MPI daemons on the desired set of nodes. MPI-based (e.g., OSPREY) jobs can now be run from the node on which the ring was created. To exit from the ring, the following command can be used:

mpdallexit

If **mpdallexit** is called, the **mpdboot** command must be executed again in order to start up MPI on the given set of nodes (these nodes need not be the same as before, so the **mpd.hosts** file can be modified).

3.2 Starting OSPREY

We will assume that a ring of MPI-ready compute nodes has already been setup according to the instructions in Sec. 3.1. From the OSPREY code directory, the program can be started using the following command:

```
mpirun -machinefile ./machines -np 5 java -Xmx1024M KStar mpi -c KStar.cfg
```

This command is parsed as follows:

- The **machines** file contains a list of nodes on which OSPREY will be executed. This list should only contain node names found in the **mpd.hosts** file (Sec. 3.1); however, a node name can appear more than once, in which case more than one job will be distributed to that node. An example **machines** file may look like this:

```
linux1  
linux1  
linux1  
linux2  
linux2  
linux3  
linux3
```

- The **-np 5** option specifies that the program should be run on five processors (so not all nodes in the example **machines** file will be used for the given execution).
- The **-Xmx1024M** option sets the maximum heap size for **java** to **1024M**. Depending on the size of the problem, the heap size may have to be increased; for some problems, a smaller value (e.g., **512M**) may be sufficient.
- The **mpi** option tells OSPREY to start a distributed computation. Note that while there are some OSPREY commands that can be executed on a single processor, all major commands require distributed execution.

- The **KStar.cfg** file is the main configuration file that specifies some basic parameters required by OSPREY. **KStar.cfg** is described in detail in Sec. 3.2.1. This file can have any filename specified by the `-c` option; for clarity, we will use the filename **KStar.cfg** throughout this documentation.

Once the **mpirun** command is executed, the following screen is displayed:

```
OSPREY Protein Redesign Software Version 1.0
Copyright (C) 2001-2009 Bruce Donald Lab, Duke University
```

```
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

```
There are additional restrictions imposed on the use and distribution
of this open-source code, including: (A) this header must be included
in any modification or extension of the code; (B) you are required to
cite our papers in any publications that use this code. The citation
for the various different modules of our software, together with a
complete list of requirements and restrictions are found in the
document license.pdf enclosed with this distribution.
```

```
OSPREY running on 5 processor(s)
```

```
>
```

NOTE: The number of processors shown on the output line `OSPREY running on 5 processor(s)` can differ between runs and should be equal to the number of processors specified when executing **mpirun**.

At this point, the program waits for the user to execute one of the OSPREY commands for protein redesign. The OSPREY commands are described in Chapter 5. All of the OSPREY commands require some standard input (e.g., input pdb structure, rotamer libraries, etc.). This standard input is described in Chapter 4.

3.2.1 Main Configuration File

This section describes the **KStar.cfg** configuration file. This file contains some parameters related to the OSPREY energy function (see Sec. 4.3), rotamer library (Sec. 4.2), and steric filter (see below). A typical file looks like this:

```
hElect true
hVDW false
hSteric false
distDepDielect true
```

```
dielectConst 6.0
vdwMult 0.95
doDihedE true
doSolvationE true
solvScale 0.8
stericThresh 0.4
softStericThresh 1.5
rotFile LovellRotamer.dat
grotFile GenericRotamers.dat
volFile AAVolumes.dat
```

The format of each line in the **KStar.cfg** file is: **parameter value**, where the **parameter** and **value** are separated by a single space. A description of the parameters is as follows:

hElect/hVDW

Determines if electrostatics/vDW energies are computed for hydrogens; both are boolean parameters. Typically, **hElect** should be set to *true*. In some cases, the user may not be as confident in the hydrogen positions in the input pdb structure, so the **hVDW** parameter can be set to *false*.

hSteric

Determines if hydrogens are used in steric checks; this is a boolean parameter. If **hSteric** is false, then steric clashes involving hydrogens are not pruned by the OSPREY steric filter.

distDepDielect

Determines if a distance-dependent dielectric should be used; this is a boolean parameter. This parameter is typically set to *true*.

dielectConst

The value of the dielectric constant. The values typically used are between 6.0 and 8.0 with a distance-dependent dielectric (**distDepDielect** set to *true*).

vdwMult

A scaling factor for the atomic vdW radii read in from the force field parameters (see Sec. 4.3).

doDihedE

Determines if side-chain dihedral energies should be computed and added to the total energy; this is a boolean parameter. This parameter is used only if side-chain dihedral flexibility is allowed - for MinDEE-based searches (see Sec. 4.3 and [10]). This parameter should typically be set to *true*.

doSolvationE

Determines if implicit solvation energies should be computed and added to the total energy; this is a boolean parameter. The EEF1 implicit pairwise solvation model [13] is used in OSPREY (see the description of the **eef1parm.dat** file in Sec. 4.3 for details).

solvScale

A multiplicative factor that scales the computed solvation energy value before adding it to the total energy of the system. Recommended values are between 0.5 and 0.8, although different values may be used depending on the type of problem (e.g., active site vs. protein surface redesign, etc.).

stericThresh

Steric overlap allowed in the initial (before minimization) steric check. If the vdW radii for a pair of atoms overlap by more than the value of this parameter, then the current rotamer-based conformation is pruned from further consideration. This parameter is used for computational efficiency. The idea is that some initial *soft* steric clash may be allowed, since conformations may minimize from such soft clashes; large clashes are not allowed and are immediately pruned. Larger values for **stericThresh** will prune fewer rotameric conformations, resulting in increased computational requirements. Smaller values for **stericThresh** may result in too much pruning and in discarding conformations that may have minimized to low energies. A value of 1.5 is typically used for the **stericThresh** parameter, with **hSteric** set to *false* (see above). If the input structure is not of high resolution, larger values (e.g., 2.0) for **stericThresh** can be used.

softStericThresh

Steric overlap allowed for a fixed rotamer conformation in BRDEE redesigns (see Sec. 5.1.3 and [8]). The value of this parameter is typically much lower than the **stericThresh** value since no energy minimization is allowed after the steric check. A typical value used for this parameter is 0.6, although lower values (≥ 0.4) may be used for very high-resolution input structures.

rotFile

The file that contains the rotamer library data for the natural amino acids (see Sec. 4.2).

grotFile

The file that contains the rotamer library data for general compounds (see Sec. 4.2).

volFile

The file that contains the rotamer volume data (see the **Mutation Search Results** discussion in Sec. 5.2.2). This file is only used for K^* redesigns. If this file does not exist, it is automatically computed.

Chapter 4

OSPREY Input Model

Performing redesigns with OSPREY requires four basic types of input: a pdb structure of the protein (or protein-ligand complex) to be redesigned (described in Sec. 4.1), rotamer libraries for natural amino acids and (optionally) for general compounds (described in Sec. 4.2), energy function parameter files (described in Sec. 4.3), and various command-dependent input configuration files that specify the different mutation search parameters (described in detail for each of the OSPREY commands in Chapter 5).

4.1 Setting up the Input Structure

After determining the protein or protein/substrate complex for redesign, a structure of that protein/complex must be used as input for OSPREY. This structure must be in the PDBv3 format, but other than that constraint, the source of the input structure (e.g., downloaded from the PDB, obtained from homology modeling, etc.) is not important. Structures using the older PDBv2.3 format can be converted to PDBv3 using programs such as the Remediator from the Richardson Lab at Duke University [12]. **NOTE:** When K^* runs are performed, two separate input structures can be used: one for the bound protein-ligand complex and one for the unbound (free) protein (see Sec. 5.2 for details). Typically, the initial input structure must be modified to make it compatible with OSPREY. Below we describe some typical modifications that are necessary for each input structure. Generally, due to the lack of standardization in the format of some input structures, the use of non-standard ligands, and the presence of certain limitations of the structure reader in OSPREY, manual per-case tweaking of the input structure may be necessary to make it compatible with the program.

Residues with Missing Atoms

OSPREY requires that no residues in the input structure have missing atoms. Since missing (heavy) atoms in crystal structures are not uncommon, one of two approaches is suggested in such cases. First, the entire residue that has missing atoms can be deleted from the input structure. Alternatively, a program such as KiNG [12] can be used to model the missing atoms in a reasonable conformation. A disadvantage of the former approach is that the flexible/mutable residues in the protein may have erroneously reduced constraint on their movement; this approach is therefore mostly applicable when the deleted residue is far from any residues that are being redesigned. A

disadvantage of the latter approach is that if the modeled residue conformation is incorrect, then the flexible/mutable residues in the protein may have erroneous constraints on their movement (too much constraint where the modeled residue is and too little constraint where the modeled residue should be). To alleviate this problem, the modeled residue may also be allowed to flex during the OSPREY mutation search in order to assume a more reasonable conformation.

Adding Hydrogens

OSPREY requires that all hydrogens be present in the input structure. The MolProbity server [4] is recommended for adding hydrogens to proteins and standard ligands that follow the PDB nomenclature. For non-standard ligands (e.g., derived chemical compounds), the Accelrys DS Visualizer program seems to perform generally well. In many cases, however, manual editing of the protonation states and hydrogen orientation may be necessary for non-standard ligands. It is generally recommended that the protonated structure be inspected for missing/misplaced hydrogens.

His Residues

His residues require special consideration. OSPREY recognizes three different protonation states for His residues:

- Both hydrogens are present for N_δ and N_ϵ . In that case, the given HIS residue must be renamed to HIP in the input structure;
- Both hydrogens are present for N_δ but only one hydrogen is present for N_ϵ . In that case, the given HIS residue must be renamed to HID in the input structure;
- Both hydrogens are present for N_ϵ but only one hydrogen is present for N_δ . In that case, the given HIS residue must be renamed to HIE in the input structure;

Steric Shell

When design is performed for proteins with more than 60-100 residues, the computational burden is significantly increased due to the increased cost of the energy minimization/computation for each candidate conformation and the increased cost of managing the data structures of the molecule. In such cases, a reduced steric shell around the flexible parts of the protein can be used, instead of all residues in the protein. The steric shell (e.g., all residues with a specified cutoff distance from any of the flexible/mutable residues or the ligand) restrains the movement of the flexible residues. The idea here is that residues that are far from the flexible/mutable residues should generally have negligible long-range energy interactions and virtually no steric interactions with the flexible/mutable residues, and can thus be excluded from the steric shell. Recommended values for the distance cutoff are 8-9 Å, although smaller values may be used depending on the size of the protein. For side-chain placement problems, the use of a steric shell should not be necessary, even for proteins with several hundred residues.

Input Structure Contents

Typically all water molecules and metal ions should be deleted from the input structure. Other than the protein residues, the input structure should only contain a ligand (if present) and a cofactor

(if present). The current version of OSPREY allows the ligand to be a natural amino acid or other small molecule. Protein-protein interactions can be modeled with certain limitations (see Sec. 6.1). Protein-peptide interactions can also be modeled with certain limitations, as long as the peptide consists only of natural amino acids (see Sec. 6.1). Cofactors can consist of multiple entities (residues or even molecules). Explicit water molecules can also be modeled as part of the rigid cofactor (Sec. 6.2). Generally, however, OSPREY is optimized for redesigning proteins and for designing protein-small molecule interactions.

Other Considerations

Several additional considerations must be taken into account when fixing the input structure for OSPREY. First, it is recommended that all TER symbols be removed from the input structure since otherwise OSPREY may interpret the *strand* (a special data structure that logically divides the molecule into a protein, ligand, and cofactor, if present) information erroneously. Second, OSPREY has no notion of chain IDs when reading the input structure, so if residues from several different chain IDs (e.g., A, B, etc.) are included, the user must make sure that each residue has a unique residue number. For example, if there are two residues in the input structure that have the same residue number (e.g., 5) but different chain IDs (e.g., A vs. B), one of the residues should be re-numbered (e.g., to 505, assuming there is no residue with that number present in the input structure). Finally, OSPREY uses atom-atom distances to determine the bond information for the molecule. Thus, the bond information for input structures with significant steric clashes can be interpreted erroneously, which can lead to problems with the energy computation (in fact, in such cases, OSPREY typically reports an error and exits the computation). It is therefore important to use good input structures with reasonable sterics.

4.2 Rotamer Libraries

Two rotamer libraries are used by OSPREY: for natural amino acids and for general compounds.

Natural Amino Acids

The default rotamers for all natural amino acids, except for Pro, are based on the modal rotamer values from the Penultimate rotamer library [17]. These rotamers are stored in the file specified by the **rotFile** parameter in the **KStar.cfg** configuration file (see Sec. 3.2.1). Different rotamer libraries for the natural amino acids can be incorporated by modifying the **rotFile** file. This file has the following format. Comment lines that are discarded by the program start with '!'. The first non-comment line in the file contains a single number representing the number of amino acids for which rotamers are defined in the file. By default, this number is 19 (all natural amino acids other than Pro; **NOTE:** mutations to/from Pro are not allowed in OSPREY, so Pro rotamers are not applicable). The remainder of the file has the following format for each amino acid type:

```
AA_name number_of_dihedrals number_of_rotamers  
dihedral_list_one_per_line  
rotamer_angles
```

The **AA_name** is given by the three-letter code for each amino acid. The **dihedral_list_one_per_line** lines give the atom names (standard PDB format) for the four atoms that form each of the **num-**

ber_of_dihedrals dihedrals for the given amino acid. Each of the **rotamer_angles** lines contains the **number_of_dihedrals** dihedral angle values for the corresponding rotamer; the total number of rotamers is given by the **number_of_rotamers** value. For example, by default, Leu has two dihedrals and five rotamers:

```
LEU 2 5
N CA CB CG
CA CB CG CD1
62 80
-177 65
-172 145
-85 65
-65 175
```

To add another Leu rotamer with rotamer angles **-85 50**, the entry for Leu must be changed to:

```
LEU 2 6
N CA CB CG
CA CB CG CD1
62 80
-177 65
-172 145
-85 65
-65 175
-85 50
```

Notice that the total number of rotamers for Leu is increased to 6 and the new rotamer angles are added at the end of the Leu rotamer list.

NOTE: If a given amino acid type has no rotamers but mutations to this amino acid are allowed, then an entry must still be present in the **rotFile** file, such that the **number_of_dihedrals** and **number_of_rotamers** values are set to 0. For example: **ALA 0 0**, or even **LYS 0 0** if no rotamers for Lys will be used (however, setting the rotamers for a natural amino acid, other than Ala or Gly, to 0 is not recommended).

General Compounds

For ligands that are not natural amino acids, rotamers must be defined in a separate file. This file is specified by the **grotFile** parameter in the **KStar.cfg** configuration file (see Sec. 3.2.1). The **grotFile** file has the same format as the **rotFile** file. The first non-comment line in the file contains a single number representing the number of compounds (non-amino acid ligands) for which rotamers are defined in the file. For each compound, the first line of information contains the three-letter code (as found in the input pdb structure), followed by the number of dihedrals and number of rotamers for that compound. Note that the number of dihedrals should reflect only the number of *flexible* dihedrals (in effect, this is the number of bonds allowed to rotate), rather than the total number of dihedrals in the compound. For each rotamer (one per line), the values of the (flexible) dihedral angles are then specified. For example, let us assume we have a ligand whose three-letter code is CHR and that has two flexible dihedrals and six rotamers; the entry for that ligand in the

grotFile may then look like this:

```
CHR 2 6
N1 C2 C3 C4
C2 C3 C4 N5
62 180
-177 68
-177 180
-90 68
-67 180
-62 -68
```

NOTE: Remember to increase the total number of compounds (the first non-comment line in the **grotFile** file) when adding the rotamer information for a new compound.

4.3 Energy Function

The energy function used in OSPREY consists of the AMBER electrostatic, van der Waals, and dihedral energy terms [21, 3] and the EEF1 implicit solvation energy term [13]. The AMBER force field parameters are read in from the following files: **parm96a.dat**, **all_amino94.in**, **all_aminont94.in**, **all_aminoct94.in**, and **all_nuc94_and_gr.in**. The EEF1 parameters are read in from the **eef1parm.dat** file. Next, we describe these parameter files in detail.

The **parm96a.dat** file

The **parm96a.dat** file contains the force field parameters for the different atom types, including the parameters used for the vdW and dihedral energy computation. **NOTE:** A distinction is made here between *atom type*, *atom name*, and *element type*. The *atom name* is the name that an atom has in the input pdb file (e.g., 'CA' for the C_α atom of a given amino acid). The *element type* is the chemical element to which the current atom corresponds (e.g., C_α 's are carbon atoms). The *atom type* refers to the force field type that is assigned to the current atom (e.g., a C_α may be assigned a 'CT' force field type according to the AMBER force field); each force field type has specific force field parameters which are used in the energy computation for a given structure.

The **parm96a.dat** file has the same format and virtually the same contents (with some minor modifications) as the **parm96.dat** file from the AMBER 9 distribution. The AMBER **parm96.dat** file is a modified version of the **parm94.dat** force field parameter file. The parameters found in the **parm96a.dat** should be sufficient for protein redesigns. In some cases, however, additional atom types (and the corresponding force field parameters) might be necessary. Additional and updated force field parameters can be incorporated into the **parm96a.dat** file, as long as the same format of the file is used. An example of how to add new force field parameters to the **parm96a.dat** file is discussed in detail below.

NOTE: All parameters from the **parm96a.dat** file are read in by OSPREY; however, the bond and angle parameters are currently not used as part of the energy function computation in OSPREY.

The `all.aminoX94.in` files

The `all.amino94.in` file contains the force field atom types and charges, as well as the atom connectivity information, for all natural amino acids. This file has the same format and virtually the same contents (with some minor modifications) as the `all.amino94.in` file from the AMBER 1994 force field [3], as found in the AMBER 9 distribution. This file should generally be left unchanged, unless a newer version of the atom types/charges (e.g., the AMBER 2002 force field) is desired. If a new version of the amino acid parameters is incorporated instead, the new `all.amino94.in` file must have the same format as the current version.

The `all.aminont94.in` and `all.aminoct94.in` files contain, respectively, the NH₃⁺ and COO⁻ amino acid atom force field types and charges. These files have the same format as the `all.amino94.in` file and are virtually the same (with some minor modifications) as the corresponding files from the AMBER 9 distribution.

The `all.nuc94_and_gr.in` file

The `all.nuc94_and_gr.in` file contains the force field atom types and charges, as well as the atom connectivity information, for: (1) nucleic acids and (2) any general compounds. The nucleic acid parameters are the same as the parameters in the `all.nuc94.in` file from the AMBER 1994 force field [3], as found in the AMBER 9 distribution. The force field parameters for general compounds can be derived using the ANTECHAMBER program [20] and added to the `all.nuc94_and_gr.in`. Next, we give an example of how to compute and add force field parameters for a general compound as part of the OSPREY input parameter files.

Example: adding force field parameters

Here, we give an example of how to add force field parameters to the OSPREY input parameter files (`parm96a.dat` and `all.nuc94_and_gr.in`) in the cases when certain parameters are missing. Let us have a small molecule ligand with three-letter name 'FCL' that differs from Phe in that chlorine is added to the para ring position (Fig. 4.1). FCL is not a natural amino acid, and force field parameters for this small molecule are not found in any of the OSPREY input files. We thus need to generate all necessary force field parameters and add them to the OSPREY input files. To do this, we will use the ANTECHAMBER program from the AMBER 9 distribution since ANTECHAMBER can generate force field parameters in the exact format required by OSPREY.

Assuming the structure of FCL is found in `fcl.pdb` (and there is nothing else in this `pdb` file), ANTECHAMBER can be run using the following command:

```
antechamber -i fcl.pdb -fi pdb -o fcl.prepi -fo prepi -c bcc -at amber
```

This generates many files, but the only output file that we need is `fcl.prepi` which contains the atom types and charges for the FCL molecule. Since ANTECHAMBER was called with `-fo prepi` and `-at amber`, the `fcl.prepi` file is in the correct format for input into OSPREY. The contents of the `fcl.prepi` may look like Fig. 4.2. All lines from `fcl.prepi`, between **This is a remark line** and **DONE** (inclusive) must be added between the last **DONE** line and the **STOP** line in the `all.nuc94_and_gr.in` file. This will allow OSPREY to read in the force field parameters for the FCL molecule.

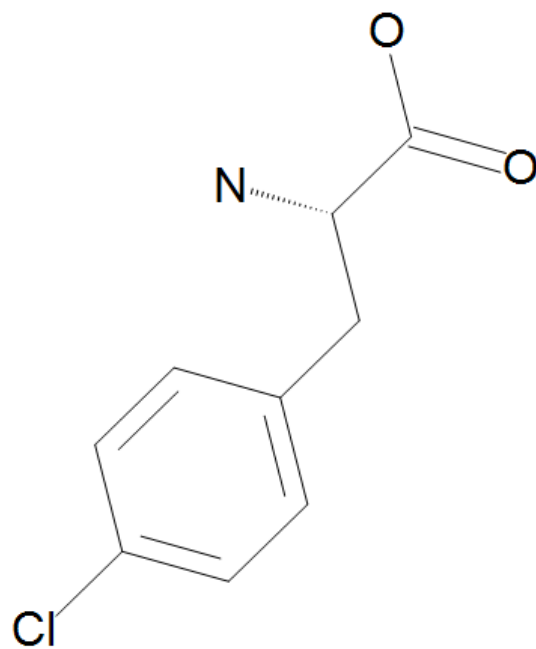


Figure 4.1: A schematic of the FCL molecule.

0 0 2

This is a remark line

molecule.res

FCL INT 0

CORRECT OMIT DU BEG

0.0000

1	DUMM	DU	M	0	-1	-2	0.000	.0	.0	.00000
2	DUMM	DU	M	1	0	-1	1.449	.0	.0	.00000
3	DUMM	DU	M	2	1	0	1.522	111.1	.0	.00000
4	O	O2	M	3	2	1	1.540	111.208	180.000	-0.70919
5	C	C	M	4	3	2	1.250	25.416	-155.202	0.93765
6	OXT	O2	E	5	4	3	1.250	124.369	-68.343	-0.80178
7	CA	CT	M	5	4	3	1.521	118.862	111.241	-0.07264
8	N	N3	3	7	5	4	1.487	109.136	-158.208	-0.82993
9	H1	H	E	8	7	5	1.070	109.492	179.981	0.41914
10	H2	H	E	8	7	5	1.070	109.510	60.002	0.42834
11	H3	H	E	8	7	5	1.070	109.521	-60.023	0.48650
12	HA	HP	E	7	5	4	1.100	109.118	82.072	0.08888
13	CB	CT	M	7	5	4	1.528	110.798	-36.499	-0.05260
14	HB2	HC	E	13	7	5	1.100	108.981	-67.828	0.06449
15	HB3	HC	E	13	7	5	1.100	108.985	48.654	0.12875
16	CG	CA	M	13	7	5	1.497	113.679	170.400	-0.11169
17	CD1	CA	M	16	13	7	1.396	120.643	-54.259	-0.12508
18	HD1	HA	E	17	16	13	1.100	119.589	0.946	0.14651
19	CE1	CA	M	17	16	13	1.399	120.862	-179.043	-0.11974
20	HE1	HA	E	19	17	16	1.101	120.141	179.898	0.15325
21	CZ	CA	M	19	17	16	1.393	119.721	-0.086	0.01562
22	C11	CL	E	21	19	17	1.770	120.084	-179.920	-0.08313
23	CE2	CA	M	21	19	17	1.399	119.809	0.048	-0.12119
24	HE2	HA	E	23	21	19	1.100	120.030	179.988	0.15163
25	CD2	CA	M	23	21	19	1.396	119.925	-0.051	-0.13198
26	HD2	HA	E	25	23	21	1.100	119.613	-179.907	0.13819

LOOP

CD2 CG

IMPROPER

CA	O	C	OXT
CD1	CD2	CG	CB
CG	CE1	CD1	HD1
CD1	CZ	CE1	HE1
CE1	CE2	CZ	C11
CZ	CD2	CE2	HE2
CG	CE2	CD2	HD2

DONE

STOP

Figure 4.2: The `fcl.prepi` file.

As is the case in this example, adding new parameters to **all_nuc94_and_gr.in** may also require adding new parameters to **parm96a.dat**. FCL has a chlorine atom, Cl. The atom type Cl is not available in the original **parm96a.dat**, and neither are any of the force field parameters related to this atom type. To obtain these parameters, the PARMCHK command from the AMBER distribution can be called:

```
parmchk -i fcl.prepi -fi prepi -o fcl.frcmod
```

This should generate the different force field parameters for the atom types from the **fcl.prepi** file, and output these parameters to the **fcl.frcmod** file. In the current example, it is sufficient to add two lines to **parm96a.dat**. First, add the line

```
Cl 35.450                               same as cl, antechamber
```

to the group of atom types and mass parameters at the beginning of **parm96a.dat**, immediately after the line

```
Cs 132.91                               cesium
```

Next, add the line

```
Cl      1.9480  0.2650                   same as cl, antechamber
```

to the group of vdW parameters at the end of **parm96a.dat**, immediately after the line

```
IB      5.0     0.1                       solvated ion for vacuum approximation
```

In some cases, it may also be necessary to add a subset of the dihedral parameters from **fcl.frcmod** to **parm96a.dat**. This will happen if an atom whose atom type is not available in the original **parm96a.dat** is also part of a dihedral whose two central atoms define a rotatable bond, as determined by the rotamers for the given molecule (see Sec. 4.2).

The eef1parm.dat file

The **eef1parm.dat** file contains solvation energy parameters for different force field atom types, as described in [13]. The mapping between amino acid atom names and force field atom types is done in the **EEF1.java** class, so any changes to the **eef1parm.dat** file should also be reflected in **EEF1.java**. The **eef1parm.dat** file only contains parameters for proteins (e.g., there are no parameters for phosphorus); cofactors and ligands that are not natural amino acids are not included in the solvation energy computation.

User control of the energy function

The user is also allowed to change certain energy function parameters, such as the value of the dielectric constant and the scaling factor for the vdW radii of the atoms, from the **KStar.cfg** configuration file. For details, see Sec. 3.2.1.

Chapter 5

OSPREY Commands

We will assume that the `mpirun` command (Sec. 3.2) has been executed and OSPREY is waiting for the user to execute one of the available commands. The OSPREY commands can be divided into three general algorithmic modules, each described in a separate section of this documentation.

Redesign can be performed using a GMEC-based approach in which candidate protein mutants are ranked based on the single best conformation (the Global Minimum Energy Conformation, GMEC) for each candidate mutant. The OSPREY GMEC-based redesign approach utilizes a number of Dead-End Elimination (DEE) algorithms combined with the A^* search algorithm for solving protein redesign problems. GMEC-based redesign with OSPREY is described in Sec. 5.1.

Alternatively, redesign can be performed using an ensemble-based approach in which candidate mutants are ranked based on an ensemble of low-energy conformations, rather than just the GMEC. The OSPREY ensemble-based approach utilizes the K^* algorithm for protein-ligand binding prediction and protein redesign. Redesign with K^* is described in Sec. 5.2.

Finally, OSPREY allows the user to use a Self-Consistent Mean Field (SCMF)-based algorithm for computing residue entropies for all residue positions in a protein. This algorithm can be used in combination with a DEE-based algorithm to predict mutations anywhere in a protein. Such a hybrid SCMF/DEE approach is applicable to, e.g., enzyme redesign for improving the target substrate specificity of the mutant enzymes. The SCMF computation in OSPREY is described in Sec. 5.3.

Once the execution of a OSPREY command completes, OSPREY exits. If more OSPREY commands must be executed, the `mpirun` command must be executed again, followed by the desired OSPREY command.

NOTE: The `mpirun` and the OSPREY commands need not be executed sequentially; rather, these commands can be executed as a single command. For example, the `doDEE` command (described in Sec. 5.1) can be executed in the following way:

```
mpirun -machinefile ./machines -np 5 java -Xmx1024M KStar mpi -c KStar.cfg doDEE System.cfg DEE.cfg
```

With this single command, the standard output from OSPREY can be redirected to a file for analysis. For example, assuming `tcsh` is used:

```
mpirun -machinefile ./machines -np 5 java -Xmx1024M KStar mpi -c KStar.cfg doDEE
```

System.cfg DEE.cfg >! logDEE.out

This command will generate a file **logDEE.out** that will store all standard output from the OSPREY run. This way, some advanced information (such as rotamer pruning and detailed running times) that is typically not generated as part of the standard OSPREY output files, will be available for analysis by the user. **NOTE:** For some problems, the standard output from a OSPREY run may generate very large files (sometimes, though rarely, exceeding 1GB), so the user should make sure that there is sufficient space in the target location.

5.1 GMEC-based Redesign

The GMEC-based redesign uses DEE-based rotamer pruning and A^* conformation enumeration. The DEE pruning stage can incorporate continuously-flexible rotamers (the MinDEE algorithm [10]) or continuous (the BD algorithm [7]) or discrete (the BRDEE algorithm [8]) protein backbones. Traditional DEE pruning [6, 18] for a rigid backbone and rigid rotamers can also be performed.

In a GMEC-based redesign, mutation sequences are ranked according to the single lowest-energy rotamer-based conformation for each sequence. Typically, either only the overall GMEC (over all mutation sequences) is identified, or a gap-free list of conformations and sequences is generated, such that all conformations/sequences within a user-specified energy window from the GMEC energy are generated by the A^* enumeration [10].

A GMEC-based mutation search can be performed by OSPREY using the following command:

```
doDEE System.cfg DEE.cfg
```

The **System.cfg** and **DEE.cfg** configuration files are described in detail below (Sec. 5.1.1). The names of the two configuration files specified after the **doDEE** command can be chosen by the user; for clarity, we will refer to these files as **System.cfg** and **DEE.cfg** throughout this documentation. The output of the **doDEE** command is also described below (Sec. 5.1.2).

Once the **doDEE** command completes its execution, the user will have a list of low-energy conformations and sequences. PDB structures for selected conformations can then be generated using the following command:

```
genStructDEE System.cfg GenStruct.cfg
```

The **System.cfg** configuration file is the same as with the **doDEE** command. The **GenStruct.cfg** file (or the corresponding user-specified filename) is described in detail in Sec. 5.1.1 below. The output of the **genStructDEE** command is described in Sec. 5.1.2 below. The **genStructDEE** command is executed on a single processor.

The GMEC-based mutation search can be applied with or without the *DACS* (Divide-And-Conquer Splitting) algorithm [9]. *DACS* divides the conformation space into non-overlapping partitions and uses partition-specific information to efficiently generate the GMEC for each partition. The partition GMEC's are then used to obtain the overall GMEC, for the full conformation space. *DACS* was found to result in speedups of up to more than three orders of magnitude when compared to DEE/ A^* runs without *DACS* [9]. *DACS* can be performed on a single processor or on a (large) cluster of processors. Generally, the user can adapt the *DACS* partitioning scheme depending on the redesign problem and the availability of computational resources.

5.1.1 Configuration Files

System.cfg

This configuration file contains the information about the system (protein) being redesigned. A typical file looks like this:

```
pdbName 1amu_8A_lig.pdb
numInAS 7
residueMap 236 239 278 299 301 322 330
```

```
pdbLigNum 550
ligAA true
numCofRes 1
cofMap 551
```

A description of the parameters is as follows:

pdbName

The name of the input pdb file of the design system. This should be the file modified by the user according to the instructions for making the input structure compatible with OSPREY, as described in Sec. 4.1.

numInAS

The number of residues modeled as flexible using rotamers and (possibly) allowed to mutate; for protein-ligand binding, this is the number of residues in the active site of the protein.

residueMap

The residue numbers of the flexible residues in the input pdb file. The total number of residue numbers must be the same as the **numInAS** value.

pdbLigNum

The residue number of the ligand in the input pdb file; -1 if no ligand.

ligAA

Is the ligand (if present) a natural amino acid (set this parameter to *true*) or some other small molecule (set to *false*)?

numCofRes

The number of cofactor residues; 0 if no cofactor. A cofactor 'residue' can in fact also be a DNA base, a water molecule, or some small molecule. Cofactor residues need not be bonded to each other.

cofMap

The residue number(s) of the cofactor in the input pdb file. The total number of residue numbers must be the same as the **numCofRes** value.

DEE.cfg

This configuration file contains the information about the DEE/ A^* mutation search parameters. A typical file looks like this:

```
runName runInfo.out
numMaxMut 7
algOption 3
```

```

doDACS true
splitFlags true
distrDACS false
doMinimize false
minimizeBB false
doBackrubs false
backrubFile none
minEnergyMatrixName 1amuArgNMPEMmin
maxEnergyMatrixName none
useEref true
initEw 2.0
pruningE 100.0
stericE 30.0
approxMinGMEC false
lambda 10000.0
preprocPairs true
pairSt 100.0
scaleInt false
maxIntScale 0.1
minRatioDiff 0.15
initDepth 1
subDepth 1
diffFact 6
genInteractionGraph false
distCutoff 10000.0
eInteractionCutoff 0.0
outputConfInfo c_1amu_
outputPruneInfo p_1amu_
ligPresent true
ligType arg
addWT true
resAllowed0 gly ala val leu ile tyr phe trp met
resAllowed1 gly ala val leu ile tyr phe trp met ser thr hip asn gln lys arg asp glu
resAllowed2 gly ala val leu ile tyr phe trp met ser thr hip asn gln lys arg asp glu
resAllowed3 gly ala val leu ile tyr phe trp met ser thr hip asn gln lys arg asp glu
resAllowed4 gly ala cys
resAllowed5 gly ala val leu ile tyr phe trp met
resAllowed6 gly ala val leu ile tyr phe trp met
resumeSearch false
resumeFilename runInfo.out.partial

```

A description of the parameters is as follows:

runName

The file in which partial results are stored. The format of this file is described in detail in Sec. 5.1.2.

numMaxMut

The maximum number of mutations from the wildtype, such that any solution generated by the algorithm will contain not more than **numMaxMut** mutations.

algOption

Determines the types of DEE criteria applied: 2-split positions DEE is used for **algOption** ≥ 2 , while DEE pairs pruning is applied for **algOption** ≥ 3 . The other DEE criteria (Bounds, simple Goldstein, 1-split position DEE) are used for any value of **algOption**. See [9] for a review of the different DEE pruning criteria.

doDACS

Determines if the algorithms should use *DACS* or not. If *false*, then *DACS* is not performed after the initial DEE pruning and the program directly proceeds to a single-processor A^* conformation enumeration; otherwise, *DACS* splitting is performed.

splitFlags

Is the split-flags technique used (see [9] for a review of this pruning algorithm)?

distrDACS

Will the *DACS* run be distributed? If *true*, then each *DACS* partition is distributed to a separate processor for evaluation. If there are more partitions than processors, a queue is formed and the distribution continues until there are no remaining partitions in the queue. If **distrDACS** is *false*, the *DACS* partitions are evaluated sequentially on a single processor.

doMinimize

Determines if energy minimization is to be performed; *true* if energy minimization is performed (for BRDEE, BD, MinDEE); *false* otherwise (for traditional DEE).

minimizeBB

Determines if backbone energy minimization is to be performed; *true* if backbone energy minimization is performed (for BD and BRDEE); *false* otherwise (for MinDEE). This parameter is taken into account only if **doMinimize** is *true*.

doBackrubs

Determines if backrubs are to be performed; *true* if backrubs are performed (for BRDEE); *false* otherwise (for BD). This parameter is taken into account only if **minimizeBB** is *true*.

backrubFile

The input file that contains the precomputed allowed backrub sets (this is the output file from the **precomputeBackrubs** command described in Sec. 5.1.3). This parameter is taken into account only if **doBackrubs** is *true*.

minEnergyMatrixName/maxEnergyMatrixName

The precomputed lower-/upper-bound pairwise energy matrix file names. For traditional DEE, where no minimization is performed, only the **min** matrix is computed and stored. These matrices are described in detail in Sec. 5.1.2.

useEref

Determines if amino acid reference energies are used as part of the energy function. The amino acid reference energies are computed using the lowest computed intra-rotamer energy for each amino acid type among all flexible residue positions (similarly to [16]).

initEw

The E_w value [10] used to guarantee that no conformations having an energy within E_w of the energy of the GMEC are pruned by the DEE algorithms. The value of **initEw** is also used as a halting condition to determine the set of conformations generated by A^* . If **initEw** is set to zero, then the only conformation that is guaranteed not to be pruned is the GMEC. Larger values of **initEw** can generate a gap-free list of multiple low-energy conformations and mutation sequences.

pruningE

Conformations with an energy lower bound greater than this value are pruned. This parameter is used by the MinBounds algorithm [9]. A value for this parameter can be generated using the **doSinglePartFn** command (Sec. 5.2) and setting **pruningE** to be the energy of the lowest-energy conformation for the wildtype protein sequence. If the value of **pruningE** is not obtained through **doSinglePartFn**, then **pruningE** can be conservatively set to a reasonably large value (e.g., 50-100 kcal/mol), so that conformations with high energy bounds are pruned.

stericE

Rotamers with intra-rotamer plus rotamer-to-template energy greater than this threshold are pruned as a preprocessing step.

approxMinGMEC

Determines if the heuristic halting condition for the DEE/ A^* computation should be used (see [1, Supporting Information, Sec. S1.2.3]); *true* if the heuristic halting condition is used; *false* if the provable halting condition (see [10, Proposition 2]) is used. The heuristic halting condition is useful for DEE algorithms that use minimization (MinDEE, BD, and BRDEE), since these algorithms require the A^* enumeration of a significantly larger number of conformations before the mutation search can be provably halted.

lambda

The cutoff for the heuristic halting condition; **lambda** is only used if **approxMinGMEC** is *true*. Let b_m be the computed lower bound on the conformational energy of the first rotameric conformation generated by A^* , and let b_c be the computed lower bound on the conformational energy of the current rotameric conformation generated by A^* . The DEE/ A^* search can then be halted when $b_c > b_m + \lambda$.

preprocPairs

Determines if rotamer pairs should be pruned based on a steric energy threshold: *true* if rotamer pairs should be pruned based on a steric energy threshold; *false* otherwise.

pairSt

If **preprocPairs** is *true*, rotamer pairs with interaction energy greater than this threshold are pruned as a preprocessing step. This parameter is analogous to the **stericE** parameter for single rotamers.

scaleInt

This parameter scales down the E_{\circlearrowleft} pairwise energy interval terms (see [7, Eq. 9]) to allow for additional rotamer pruning. It is recommended to keep this parameter set to *false* since the effects of applying interval scaling are not yet fully evaluated.

maxIntScale

The maximum scaling factor (0-1) for the interval terms (if **scaleInt** is *true*). The value of this scaling factor decreases as a function of the distance between the respective residue positions involved for each interval term.

minRatioDiff

This is a parameter that is used for choosing a major split position for the *DACS* algorithm, in combination with the original *p*-ratio approach. The higher the value, the more favored lower-numbered residue positions are (see [9] for details). Faster run times were achieved with a value of 0.15 as compared to a value of 0.0, although optimization of this parameter has not been attempted.

initDepth

This parameter is only used if **distrDACS** is *true*. **initDepth** determines the number of *major* splitting positions for the *DACS* algorithm. *DACS* partitions are formed by enumerating all combinations of unpruned rotamers for each residue position (each combination is given by a single rotamer choice for each residue position). For example, if **initDepth** is 2 and each of the two selected major split positions has q rotamers, then there will be q^2 *DACS* partitions. Each partition is then distributed to a separate processor for evaluation. Harder problems may require larger values of **initDepth** (depending on the processor availability). However, if the value of **initDepth** is too large (e.g., close to **numInAS**, the total number of flexible residue positions), the partition enumeration may come at a significant additional computational cost. The **initDepth** partitioning will be referred to as *major partitioning*.

subDepth

This parameter is used when **doDACS** is *true* and (1) **distrDACS** is *false*, or (2) **distrDACS** is *true* and the current major partition has already been distributed for computation to a given processor. The value of **subDepth** gives the number of (additional) *minor* splitting positions. Partitions (rotamer combinations) are formed the same way as with **initDepth**. The difference from **initDepth** is that the newly-formed partitions are evaluated sequentially

on the current processor, and not distributed to separate processors. If the sum of **initDepth** and **subDepth** is too large (e.g., close to **numInAS**), the partition enumeration may come at a significant additional computational cost. The **subDepth** partitioning will be referred to as *sub-partitioning*.

diffFact

Consider a given major partition for evaluation (or the single initial partition, if **distrDACS** is *false*). Also, note that sub-partitioning is implemented as a recursive procedure in OSPREY. Let $i < \mathbf{subDepth}$ be the current residue position, such that the current sub-partition (combination of rotamers) is only assigned for i of the minor splitting positions. If the total number of remaining unpruned conformations for the current partially-assigned partition is not more than $10^{\mathbf{diffFact}}$, then the sub-partitioning is stopped and A^* enumeration for the current partially-assigned partition is executed. Otherwise, the sub-partitioning is continued until either the **diffFact** stopping condition is reached, or all minor splitting positions are assigned, at which point the A^* enumeration is executed.

genInteractionGraph, distCutoff, eInteractionCutoff

The **genInteractionGraph** parameter must be set to *false*. **genInteractionGraph**, **distCutoff**, and **eInteractionCutoff** are used for residue interaction graph generation, rather than conformation/sequence generation, and are thus not applicable here.

outputConfInfo

The output file name that will store information about the best-energy conformations found. If **distrDACS** is *true*, a directory named **conf_info** must exist in the source code directory, since this is where the output files will be stored. These output files are discussed in detail in Sec. 5.1.2.

outputPruneInfo

The output file name that will store information about the pruning done. If **distrDACS** is *true*, a directory named **conf_info** must exist in the source code directory, since this is where the output files will be stored. These output files are discussed in detail in Sec. 5.1.2.

ligPresent

Will the ligand (if at all present in the input pdb structure) be used in the DEE/ A^* search: *true*, if the ligand is present and will be used; *false* otherwise. Note that this parameter has a different meaning from the **pdbLigNum** in the **System.cfg** file. **pdbLigNum** determines if a ligand is present in the input structure (and if so, what the corresponding pdb residue number is). However, if **pdbLigNum** is *true* but **ligPresent** is *false*, the ligand will be removed from the structure and will not be used in the DEE/ A^* conformation search (i.e., only the protein and the cofactor, if present, will be evaluated).

ligType

The type of the ligand, if **ligPresent** is *true* (anything otherwise).

addWT

Determines if the redesign positions should be allowed to keep their wildtype identity, in addition to allowing them to mutate to the amino acid types in **resAllowed*i*** (see below).

resAllowed*i*

What amino acid types are allowed for the *i*-th flexible residue (numbered from 0, 1, ..., **numInAS**)? Use standard three-letter amino acid codes separated by a single space. If for a given residue position *j*, only the wildtype should be allowed, set **addWT** to *true*, and leave **resAllowed*j*** empty (however, a single empty space must still be present immediately after **resAllowed*j***).

resumeSearch

Is the current run an unfinished search, so that the resume files must be loaded? This is used only to resume a distributed *DACS* run, so that the completed partitions will not have to be computed again. The resume capability is discussed in detail in Sec. 5.1.2.

resumeFileName

What file contains the resume information?

GenStruct.cfg

This configuration file contains the information about the structure generation parameters. A typical file looks like this:

```
confResFile conf_info.sel
numResults 5
outputPDBs true
doMinimize false
minimizeBB false
doBackrubs false
backrubFile none
ligPresent true
ligType arg
```

The following parameters must have the same value as the respective parameters in the **DEE.cfg** file:

```
doMinimize
minimizeBB
doBackrubs
backrubFile
ligPresent
ligType
```

The remaining parameters are described as follows:

confResFile

The file that contains the input information, one conformation per line. This file is obtained from the output of the **doDEE** command. The format of this file is described in detail in Sec. 5.1.2.

numResults

The number of lines in the **confResFile** file.

outputPDBs

Determines if pdb structures will be generated for each conformation from the **confResFile** file. This parameter should be set to *true*.

5.1.2 Output Files

Pairwise Energy Matrices

The **doDEE** command first computes a lower-bound and an upper-bound pairwise energy matrix (in the case of MinDEE, BD, and BRDEE) or a single matrix (in the case of traditional DEE). These matrices contain the computed interaction energies for the protein template (the protein backbone and the side-chains not modeled as flexible), between each rotamer for each residue position and the protein template, and between each pair of rotamers for the different pairs of residue positions. Only the energies for the amino acid types (and the corresponding rotamers) specified by the **resAllowed*i*** parameters are computed. For example, in the **DEE.cfg** sample file shown in Sec. 5.1.1, **resAllowed4** has **gly ala cys** as the allowed amino acid types (we will assume that the wildtype identity at that residue position is Ala). In that case, energies involving, e.g., Leu rotamers at that residue position will not be computed.

In the case of MinDEE, BD, and BRDEE, energy minimization (either rotamer or backbone) is allowed. As a result, the energies between rotamer pairs (and between rotamers and the protein template) are not rigid and will depend on the rotamer identities and conformation of the surrounding residues (see [10, Fig. 2]). If the energy minimization of the rotamers or the protein backbone is restrained (by, e.g., restraints on the side-chain dihedral movement), then both a lower-bound and an upper-bound for each pairwise energy can be computed within the specified restraints. Hence, the computed lower/upper energy bounds are saved into two matrices. These matrices are saved as binary files, as specified by the **minEnergyMatrixName**/**maxEnergyMatrixName** parameters in the **DEE.cfg** file. The two energy matrices are used by MinDEE, BD, and BRDEE during the respective DEE pruning stage. The lower-bound matrix is further used during the A^* enumeration stage to generate conformations in order of increasing lower bounds on their energies [10]. For traditional DEE, a single matrix (specified by the **minEnergyMatrixName** parameter) is computed and saved since energy minimization is not allowed. Once the energy matrices are computed, **doDEE** can read them in every time the DEE/ A^* search is restarted, and the computation of these matrices need not be repeated, as long as the input structure and allowed mutations (specified by the **resAllowed*i*** parameters) remain unchanged.

Mutation Search Results

After the computation of the pairwise energy matrices is done, the program moves to the DEE pruning stage, followed by the A^* enumeration stage. If the *DACS* algorithm is applied (**doDACS** in **DEE.cfg** is set to *true*), DEE pruning is first applied for the entire conformation space, until no more dead-ending rotamers/pairs can be identified. The pruning and enumeration stages are then applied separately for each of the *DACS* partitions. Depending on whether *DACS* is applied and whether *DACS* is performed as a distributed run, different files are generated during the mutation search.

Mutation search on a single processor (with or without *DACS*)

Conformation File. Two different types of mutation search are discussed here: DEE/ A^* without *DACS* (**doDACS** is *false*), and *DACS* on a single processor (**doDACS** is *true* and **distrDACS** is *false*). In these cases, one conformation output file, specified by the **outputConfInfo** parameter in **DEE.cfg** is generated during the mutation search. This file contains all conformations generated by A^* and within the specified energy window (the **initEw** parameter) from the energy of the respective GMEC. **NOTE:** No structures are saved during this part of the program execution; rather, only the conformation data necessary to generate the structures is saved, along with the respective computed conformational energies.

Each line in the conformation file corresponds to a single rotamer-based conformation generated by A^* . The format of each line is as follows:

```
c a1 a2 ... an aL r1 r2 ... rn rL unMinE: eu minE: em minBound: emb bestE: eb
```

Here, **c** is the conformation number, in the order generated by A^* ; if *DACS* is applied, conformation numbers are partition-specific, so there might be multiple conformations with the same conformation number (but that are part of different partitions). **a_k** is the corresponding three-letter amino acid name for the k^{th} of the total of n (where $n = \mathbf{numInAS}$) flexible residue positions; **a_L** is the three-letter name of the ligand (if present). Similarly, **r_k** is the corresponding *rotamer number* for the k^{th} flexible residue position; **r_L** is the rotamer number of the ligand (if present). A *rotamer number* is determined based on the index (starting from 0) of the given rotamer for the given amino acid into the input rotamer library. For example, rotamer 3 of Leu corresponds to χ angles $-85\ 65$ in the **LovellRotamer.dat** rotamer library (see Sec. 4.2). The value **e_u** represents the conformational energy before minimization (the energy of the conformation for a rigid backbone and rigid rotamers). The value **e_m** represents the conformational energy after the respective type of minimization (backbone or side-chain); if no minimization is allowed, **e_m** = **e_u**. ‘minBound: **e_{mb}**’ is only present in the output if energy minimization is allowed and represents the computed lower energy bound for the current conformation; this lower energy bound is used by A^* for enumerating conformations in order. The value **e_b** is the best (lowest) conformational energy found in the A^* search so far. An example (partial) **outputConfInfo** file (for seven flexible residue positions, an Arg ligand, single-processor *DACS*, and no minimization) may look like this (lines are wrapped):

```
1 MET GLY ASP ARG ALA ALA MET ARG 6 0 2 18 0 0 9 31 unMinE: -273.75903 minE:
-273.75903 bestE: -273.75903
2 MET GLY ASP MET ALA ALA MET ARG 6 0 2 6 0 0 9 31 unMinE: -271.96558 minE:
-271.96558 bestE: -273.75903
3 MET GLY ASP ARG ALA ALA GLY ARG 6 0 2 18 0 0 0 31 unMinE: -271.7832 minE:
```

-271.7832 bestE: -273.75903

1 MET ASP SER ARG GLY ALA VAL ARG 6 3 2 18 0 0 1 29 unMinE: -276.5042 minE: -276.5042 bestE: -276.5042

2 MET ASP SER ARG GLY ALA VAL ARG 6 3 1 18 0 0 1 29 unMinE: -276.4287 minE: -276.4287 bestE: -276.5042

NOTE: In some cases, the conformations stored in the **outputConfInfo** file can be a superset of the conformations actually within **initEw** of the GMEC:

- If energy minimization is allowed (i.e., MinDEE, BD, or BRDEE), the set of A^* -generated conformations may be significantly larger than the set of conformations actually within **initEw** from the GMEC. This is due to the fact that, in the cases with energy minimization, A^* enumerates conformations in order of increasing *lower bounds* on their energies. However, since different conformations can minimize differently, the order of the lower energy bounds is not necessarily the same as the order of the conformations when the *actual minimized* energies are taken into account [10].
- If *DACS* is used, then additional conformations whose energy lower bounds are within **initEw** of the partition-specific GMEC (but not necessarily of the overall GMEC) may also be generated and stored.

NOTE: If **approxMinGMEC** in **DEE.cfg** is set to *false*, then only the conformations with *actual* energies within **initEw** of the GMEC (or the partition-specific GMEC) are stored, although a larger number of conformations may be generated by A^* . If **approxMinGMEC** is set to *true*, then all conformations generated by A^* are stored, independent of the actual energies. Since **approxMinGMEC** is also used as a heuristic halting condition for the DEE/ A^* search (see Sec. 5.1.1), requiring both that all A^* -generated conformations be saved and the provable halting condition be used, will also require that the **lambda** parameter in **DEE.cfg** be set to some very large value (e.g., 10000000).

After the mutation search is complete, the conformations in the **outputConfInfo** file can be sorted in order of their e_m (minimized) energies. The conformation with the lowest e_m energy will be the respective GMEC for the given problem. All (or a subset of the) conformations within **initEw** of the GMEC energy can then be extracted for further analysis and structure generation (see below).

Pruning File. In addition to the **outputConfInfo** conformation file, a single-processor *DACS* run outputs an **outputPruneInfo** pruning information file. The **outputPruneInfo** file stores information about the partition-specific conformation pruning achieved by *DACS*. This file is useful for comparing the pruning efficiency of different algorithms, as described in [9]. The information in this file is, however, not directly related to conformation and mutation scoring and ranking.

Mutation search with distributed *DACS*

When distributed *DACS* is performed (**doDACS** and **distrDACS** are *true*), one conformation file and one pruning information file are stored for each partition that is evaluated on a separate processor. For each partition, these two files are saved in a directory named **conf_info**; the names of each such pair of files are obtained by concatenating, respectively, the **outputConfInfo** and **outputPruneInfo** values with a unique partition-specific index. The format of the **outputConfInfo**

and **outputPruneInfo** files is the same as with the single-processor mutation search described above. When the computation for all partitions is done, all of the **outputConfInfo** files can be concatenated and the conformations can be sorted in order of increasing e_m energy. The conformation with the lowest e_m energy will be the respective GMEC for the given problem. All (or a subset of the) conformations within **initEw** of the GMEC energy can then be extracted for further analysis and structure generation (see below).

NOTE: One temporary file (filename **rot_outXXX**, where **XXX** is an index unique to each program execution) that is used for communication between the main node and the work nodes is also output just before the start of the partition distribution. This file is automatically deleted when the computation for all partitions is completed.

NOTE: A secondary thread is run throughout the mutation search to allow communication between partitions. This allows updating all partitions when a new best energy is found by one of these partitions. This way, partitions with high-energy partition-specific GMEC's can provably halt the computation before enumerating all conformations within **initEw** of the partition-specific GMEC. For efficiency, the secondary thread is only run every few minutes, so the updates to the different partitions can be delayed.

Resuming a mutation search. When distributed *DACS* is performed, an interrupted mutation search can be resumed, so that only partitions for which the computation did not complete are re-distributed for evaluation. When a processor completes the computation for a given partition, the partition information is output to the file specified by the **runName** parameter in the **DEE.cfg** file. If a mutation search is interrupted (e.g., if some work nodes crash) before the completion of the computation for all partitions, the partial results in the **runName** file can be copied over to the file specified by the **resumeFileName** parameter in **DEE.cfg**. The **outputConfInfo** and **outputPruneInfo** files in the **conf.info** directory must be moved to a backup directory. The **resumeSearch** parameter in **DEE.cfg** should then be set to *true*, and the **doDEE** command should be executed again. The distributed *DACS* computation then reads in the already-computed partitions from the **resumeFileName** file, and restarts the computation only for the remaining partitions. When the computation for all of the partitions is done, the **outputConfInfo** and **outputPruneInfo** files generated after the resume (these will all be in the **conf.info** directory) must be moved to the backup directory where all **outputConfInfo** and **outputPruneInfo** files before the resume were stored. This copy should overwrite all partial files in the backup directory (corresponding to partitions for which the computation started but did not complete before the resume) with the corresponding complete files. The **outputConfInfo** and **outputPruneInfo** files already completed before the resume will not be modified since each partition corresponds to files with unique filenames that are independent of whether, and how many times, the search is resumed (as long as the search parameters remain unchanged). Finally, all of the completed **outputConfInfo** files should be concatenated, and the conformations should be sorted and ranked according to their e_m energy.

Structure Generation

The computed structures for each of the total of **numResults** conformations from the input **confResFile** file (both parameters are from **GenStruct.cfg**) are output to a directory named **pdb**s. The filename of each structure is a concatenation of **saveMol** and the line number of the corresponding conformation in the input **confResFile** file. **NOTE:** Each of the generated structures

contains a **REMARK 7** line that gives the computed (minimized) energy for that structure. However, this energy does not include amino acid reference energies. This energy may thus differ from the energy output to the corresponding **outputConfInfo** file if the **useEref** flag is set to *true* during the **doDEE** execution.

5.1.3 Performing Backrubs

To perform a mutation search using backrubs, an additional OSPREY command must be executed before the execution of **doDEE** in order to generate a list of allowed backrubs for the selected flexible residue positions. The list of backrubs can be generated with the following command:

```
precomputeBackrubs System.cfg n s br.in
```

The **System.cfg** file is as described in Sec. 5.1.1 and must have the same parameter values as in the subsequent **doDEE** call. The number n determines the number of backrub steps in each direction for the primary rotation of each flexible residue (see [8]). The number s determines the backrub step size. The last argument (in this example, **br.in**) determines which file will store the computed list of backrubs. This file will be used as input for the **doDEE** execution and must thus have the same filename as the **backrubFile** parameter in **DEE.cfg** (see Sec. 5.1.1). An example call to **precomputeBackrubs** is as follows:

```
precomputeBackrubs System.cfg 2 4.0 br.in
```

In this example, a list of 5 primary backrub rotations will be generated for each flexible residue, with values -8.0 , -4.0 , 0.0 , 4.0 , and 8.0 degrees. The backrub generation algorithm then computes the two peptide rotations for each primary rotation using the approach described in [8]. Finally, a steric and τ -angle filter is applied to prune sets of the candidate backrubs [8]. Backrubs that pass the pruning filters are output to the file specified by the last argument in the **precomputeBackrubs** call. The format of that file is as follows. The first line in that file contains two numbers: (1) the number of flexible residue positions (equal to the value of the **numInAS** parameter in **System.cfg**), and (2) the number of backrubs per residue position (equal to $2 \times n + 1$, where n is the second argument in the **precomputeBackrubs** call). Each of the remaining lines corresponds to a single combination of primary and flanking backrub rotation angles for each residue position (there can be multiple backrub combinations per residue position). The format of each of these lines is the following:

```
i k  $\theta_c$   $\theta_l$   $\theta_r$ 
```

A description of the data is as follows:

- i : the index of the current residue number into the flexible-residue array. For example, if the flexible residue numbers are **236 239 278 299 301 322 330**, then the index of 236 will be 0, while the index of 322 will be 5. This index is automatically generated by the program and is used when the backrub file is read by the **doDEE** command.
- k : the index of the current backrub. This index is generated automatically and has a range $0 \dots 2 * n$, where n is the second argument in the **precomputeBackrubs** call.
- θ_c : the value of the current primary rotation angle.

- θ_l, θ_r : the values of the left and right flanking rotation angles, respectively. These rotation angles are computed as a function of the corresponding primary rotation angle.

An example file generated by `precomputeBackrubs` is shown here:

```

7 5
0 2 0.0 0.0 0.0
0 3 4.0 -1.3422085 -3.6416507
0 4 8.0 -2.6781783 -7.0035505
1 0 -8.0 2.8148565 7.038195
1 1 -4.0 1.4027587 3.3622477
1 2 0.0 0.0 0.0
2 2 0.0 0.0 0.0
2 3 4.0 -2.3724327 -2.0907638
2 4 8.0 -4.670408 -4.273453
3 1 -4.0 2.5223923 1.8658059
3 2 0.0 0.0 0.0
3 3 4.0 -2.4386685 -1.9066759
3 4 8.0 -4.8066154 -3.8650274
4 0 -8.0 6.8834524 3.2620065
4 1 -4.0 3.4216287 1.630518
4 2 0.0 0.0 0.0
4 3 4.0 -3.3753068 -1.6301426
5 2 0.0 0.0 0.0
5 3 4.0 -2.352193 -2.1507177
5 4 8.0 -4.6454926 -4.3780875
6 2 0.0 0.0 0.0
6 3 4.0 -1.0102168 -3.5866969
6 4 8.0 -1.9774139 -6.9551406

```

Once the list of backrubs is computed and saved, the `doDEE` command can be executed.

NOTE: In some cases, the user may decide to manually edit the backrub file. In the example file above, adding one more backrub with angles **8.0 -10.0 -5.0** for the residue with index **5**, requires that: (1) the first line in the file be changed to **'7 6'**, and (2) the following line be added immediately after the last line for residue index 5 (starting with **'5 4'**) and before the first line for residue index 6 (starting with **'6 2'**):

```
5 5 8.0 -10.0 -5.0
```

NOTE: The second number on the first line in the file must only be increased by the maximum number of added backrubs for any given residue position. For example, if the user adds 2, 3, 2, 1, 0, 1, and 0 backrubs respectively for the seven flexible residues in the example above, then the first line in the backrub file must be modified to **'7 8'**.

NOTE: Deleting lines from the backrub file is also allowed, as long as the line in which the three rotation angles are 0.0 (corresponding to the initial backbone conformation) is kept for each of the flexible residue positions.

NOTE: The `precomputeBackrubs` command can also be executed before the `KSMaster` command for K^* redesign (Sec. 5.2) with backrubs.

5.2 Redesign Using K^*

The K^* algorithm can be used for protein-ligand binding prediction in protein redesign, with applications to enzyme redesign and small-molecule inhibitor design. K^* requires that an initial structure of the bound protein-ligand complex exists. K^* differs from typical DEE/ A^* mutation searches (e.g., see Sec. 5.1) in several ways:

1. K^* explicitly models the bound protein-ligand complex vs. the unbound (free) protein and ligand;
2. K^* computes partition functions over ensembles of conformations. This is in contrast to GMEC-based algorithms where each mutant is scored and ranked based on the corresponding single lowest-energy conformation (the GMEC for that mutant);
3. Energy minimization is performed for each of the conformations part of the K^* ensemble.

K^* is a provably-accurate algorithm with respect to the input model (the input structure, the rotamer library, and the energy function) used. Enumerating all conformations for a given protein-ligand complex is computationally infeasible. K^* thus computes a provably-accurate approximation to the binding constant for each candidate protein-ligand complex by evaluating only a very small portion of candidate low-energy conformations; the contribution of the remaining conformations is provably guaranteed to be less than the approximation accuracy. The accuracy of the binding constant approximation is determined by the user. In computational tests, K^* was shown to be able to enumerate less than 0.5% of all rotamer-based conformations for a given protein-ligand complex, while at the same time guaranteeing that the computed partial partition function was at least 97% of the full partition function (when all rotamer-based conformations are included) [10]. Details of the K^* algorithm can be found in [10, 1].

The basic flow of the K^* computation is as follows:

1. First, a set of residue positions in the input structure is selected for mutation. Typically, these residue positions are part of the protein binding/active site and are in direct contact with the ligand.
2. A set of allowed amino acid mutations is then selected for each of the mutable residue positions.
3. Since the K^* score computation is performed separately for each mutant, it is recommended that the total number of candidate mutants be limited (currently, up to several tens of thousands of mutants can be evaluated for K^* in a reasonable time on a large cluster of processors). This can be achieved by allowing only k -point mutation sequences, such that any k of the mutable positions are allowed to mutate at the same time, while all remaining mutable positions are modeled as flexible using rotamers. All combinations of k -point mutation sequences can be generated by K^* .
4. The list of k -point mutation sequences is then input into the K^* volume filter, so that under- and over-packed sequences (relative to the wildtype sequence and wildtype ligand) are pruned. Only sequences that pass the volume filter are evaluated further.

5. For each sequence that passes the volume filter, a provably-accurate K^* approximation score is computed. For each K^* score computation, partition functions for the bound protein-ligand complex and the free protein and free ligand are computed. For the partition function computation, DEE is applied first as a pre-processing step to prune candidate rotamers (and hence, conformations), followed by the A^* search to enumerate an ensemble of low-energy conformations. For computational efficiency, an *inter-mutation pruning filter* can be used as part of K^* , so that the requirement for provable score accuracy is not enforced for low-scoring sequences (mutation sequences with higher K^* score are predicted to be better binders).
6. The candidate mutants are then ranked in order of decreasing K^* scores. A set of high-scoring mutants (e.g., the top forty mutants) is finally selected for further analysis and (possibly) structure generation.

A K^* mutation search can be performed by OSPREY using the following command:

KSMaster System.cfg MutSearch.cfg

The **System.cfg** configuration file is as described in Sec. 5.1.1. The **MutSearch.cfg** file is described in detail below (Sec. 5.2.1). The names of the two configuration files specified after the **KSMaster** command can be chosen by the user; for clarity, we will refer to these files as **System.cfg** and **MutSearch.cfg** throughout this documentation. The output of the **KSMaster** command is also described below (Sec. 5.2.2).

Once the K^* score computation is complete, structures for a given mutant can be generated using the following command:

doSinglePartFn System.cfg SinglePF.cfg

The **System.cfg** configuration file is the same as with the **KSMaster** command. The **SinglePF.cfg** file (or the corresponding user-specified filename) is described in detail in Sec. 5.2.1 below. In effect, the **doSinglePartFn** command performs a partition function computation (either for the bound or unbound state) for a given single mutation sequence. The output of the **doSinglePartFn** command is described in Sec. 5.2.2 below. The **doSinglePartFn** command is executed on a single processor.

5.2.1 Configuration Files

MutSearch.cfg

```
runName 1amuArgMutSearch
mutFileName 1amuArg2MUT.mut
numMutations 2
targetVolume 620.0
volumeWindow 40.0
doMinimize true
minimizeBB false
doBackrubs false
backrubFile none
minEnergyMatrixName 1amuArgSCPEMmin
```



```

maxEnergyMatrixName 1amuArgSCPEMmax
initEw 6.0
pruningE 100.0
stericE 30.0
scaleInt false
maxIntScale 1.0
epsilon 0.03
gamma 0.01
repeatSearch true
useUnboundStruct true
unboundPdbName 1amu_8A_lig_UB.pdb
minEnergyMatrixNameUnbound 1amuArgSCPEMminUB
maxEnergyMatrixNameUnbound 1amuArgSCPEMmaxUB
ligPresent true
ligType arg
addWT true
resAllowed0 gly ala val leu ile tyr phe trp met
resAllowed1 gly ala val leu ile tyr phe trp met ser thr hip asn gln lys arg asp glu
resAllowed2 gly ala val leu ile tyr phe trp met ser thr hip asn gln lys arg asp glu
resAllowed3 gly ala val leu ile tyr phe trp met ser thr hip asn gln lys arg asp glu
resAllowed4 gly ala cys
resAllowed5 gly ala val leu ile tyr phe trp met
resAllowed6 gly ala val leu ile tyr phe trp met
resumeSearch false
resumeFileName 1amuArgMutSearch.partial

```

The following parameters are as described for the **DEE.cfg** file in Sec. 5.1.1:

```

doMinimize
minimizeBB
doBackrubs
backrubFile
minEnergyMatrixName
maxEnergyMatrixName
initEw
pruningE
stericE
scaleInt
maxIntScale
ligType
addWT
resAllowed

```

A description of the remaining parameters is as follows:

runName

The file in which partial results are stored. The format of this file is described in detail in Sec. 5.2.2.

mutFileName

The file in which all candidate mutation sequences are stored. If not available, this file is computed and stored dynamically using the **targetVolume**, **volumeWindow**, **numMutations**, and **resAllowed***i* parameters described below. The format of this file is described in detail in Sec. 5.2.2.

numMutations

The **numMutations** parameter is used for generating the **mutFileName** file; if the **mutFileName** file has already been generated, this parameter is ignored. This parameter determines the maximum number of mutations from the wildtype, such that only up to **numMutations**-point sequences are considered. In a k -point mutation search, any k of the n flexible residue positions are allowed to simultaneously mutate, while the remaining flexible residue positions are allowed to change their side-chain conformation (but not their amino acid identity).

targetVolume

The **targetVolume** parameter is used for generating the **mutFileName** file; if the **mutFileName** file has already been generated, this parameter is ignored. This parameter determines the target volume for the active site used by the K^* volume filter (see [10]). This target volume is computed as the sum of the volumes for the active site residues (taken from the **volFile** file specified in **KStar.cfg**), plus the difference between the volumes for the wildtype ligand and the target ligand. This parameter can only be used for amino acid ligands. If the ligand is not a natural amino acid, it is recommended that the volume filter be switched off (see Sec. 5.2.2 for details).

volumeWindow

The **volumeWindow** parameter is used for generating the **mutFileName** file; if the **mutFileName** file has already been generated, this parameter is ignored. This parameter is used by the volume filter to determine which mutation sequences with the target ligand are under- or over-packed relative to the wildtype sequence/ligand. Only mutation sequences with volumes within **volumeWindow** around the **targetVolume** are kept for further evaluation. As with **targetVolume**, this parameter can only be used for amino acid ligands. If the ligand is not a natural amino acid, it is recommended that the volume filter be switched off (see Sec. 5.2.2 for details).

epsilon

The K^* partition function approximation parameter (see [10] for details). This parameter guarantees that the computed partial partition functions will be at least $(1 - \epsilon)q$, where q is the full partition function (when all rotamer-based conformations are considered). The value of this parameter should be between 0 and 1; for example, a value of 0.03 corresponds to a $\geq 97\%$ approximation accuracy.

gamma

The inter-mutation approximation parameter(see [10] for details). This parameter determines for which mutation sequences a provably-accurate approximation should be computed. The value of this parameter should be between 0 and 1. A value of 0 guarantees an ε -approximation for all candidate sequences; a value of 1 guarantees an ε -approximation only for the top-scoring (best) sequence; a value of 0.01 guarantees an ε -approximation for all sequences whose scores are within two orders of magnitude from the top-scoring sequence. The idea is that, in general, only sequences close to the top sequences will be of interest since they are predicted to be good binders. A provably-good approximation is thus required for such sequences. All other sequences are guaranteed to have low scores, so the partition function/score computation can be halted early, without achieving provable guarantees. **NOTE:** The use of the inter-mutation filter can result in a significant speedup. In some cases, however, provably-accurate approximations may be required even for low-scoring sequences (e.g., in negative design where it is important to be able to correctly predict poor interactions); in such cases, the inter-mutation filter can be turned off by setting the **gamma** parameter to 0.

repeatSearch

If at the end of the A^* conformation enumeration for a given partition function computation the ε -approximation cannot be guaranteed, should the pruning/enumeration be repeated? The computation is repeated at most one (see [10] for details). It is recommended to keep this parameter set to *true*.

useUnboundStruct

Determines if a different unbound structure is used for the unbound partition function computation. By default, K^* uses the input structure for the bound protein-ligand complex (specified by the **pdbName** parameter in the **System.cfg** file described in Sec. 5.1.1) for both the bound and unbound (free protein) partition function computation. In the default case, the unbound (free) protein structure is obtained by simply removing the ligand from the input structure. This approach is useful if only a bound protein-ligand structure is available. In cases where a structure of the free protein is also available, that structure can be used for the unbound partition function computation; the **useUnboundStruct** parameter should then be set to *true*. **NOTE:** If both the bound and unbound structures are used, the only difference between these structures should be the conformation of the residues in these structures and the lack of a ligand in the unbound structure; the two structures must have the same input residues (e.g., if residue 278 is present in the bound structure, it must also be present in the unbound structure, and vice versa).

unboundPdbName

The name of the input pdb file for the unbound (free) protein. This should be the file modified by the user according to the instructions for making the input structure compatible with OSPREY, as described in Sec. 4.1. This parameter is only taken into account if **useUnboundStruct** is *true*.

minEnergyMatrixNameUnbound/maxEnergyMatrixNameUnbound

The precomputed lower-/upper-bound pairwise energy matrix file names for the unbound (free) protein. These matrices are analogous to the **minEnergyMatrixName** and **maxEnergyMatrixName** matrices, which are used for the bound protein-ligand computation (or for both the bound and unbound computation if **unboundPdbName** is *false*). The **minEnergyMatrixNameUnbound** and **maxEnergyMatrixNameUnbound** parameters are only taken into account if **unboundPdbName** is *true*.

resumeSearch

Is the current run an unfinished search, so that the resume files must be loaded and the completed sequence scores will not have to be computed again. The resume capability is discussed in detail in Sec. 5.2.2.

resumeFileName

What file contains the resume information?

SinglePF.cfg

```
doMinimize true
minimizeBB false
doBackrubs false
backrubFile none
minEnergyMatrixName 1amuArgSCPEMmin
maxEnergyMatrixName 1amuArgSCPEMmax
initEw 6.0
pruningE 100.0
stericE 30.0
scaleInt false
maxIntScale 1.0
epsilon 0.03
gamma 0.01
repeatSearch true
resMut ALA TRP ASP ILE GLY ALA ILE
ligPresent true
ligType arg
outputPDBs true
pdbPrefix pdb/1amuArg_
bestScore 23500089
protPartFn 739352
```

The following parameters are as described for the **MutSearch.cfg** file and must have the same values:

```
doMinimize
minimizeBB
doBackrubs
backrubFile
```

minEnergyMatrixName
maxEnergyMatrixName
initEw
pruningE
stericE
scaleInt
maxIntScale
epsilon
gamma
repeatSearch

A description of the remaining parameters is as follows:

resMut

Determines for which mutation sequence the partition function computation should be performed. The value of this parameter should be a string of three-letter amino acid codes that specify the amino acid type for each of the **numInAS** mutable residue positions (see Sec. 5.1.1).

ligPresent

Determines if **doSinglePartFn** will be performed for the bound state partition function computation (**ligPresent** is *true*) or for the unbound state partition function computation (**ligPresent** is *false*).

ligType

The type of the ligand; if **ligPresent** is *true*, then **ligType** must have the same value as in **MutSearch.cfg**.

outputPDBs

Determines if pdb structures for the conformations from the computed partition function ensemble should be generated. Set this parameter to *true*.

pdbPrefix

The filename prefix of the pdb structures. It is recommended to store these structures in a separate subfolder since in some cases the number of generated structures can be very large. This parameter is only taken into account if **outputPDBs** is *true*.

bestScore

This parameter is only taken into account for the bound partition function computation (i.e., when **ligPresent** is *true*). The value of this parameter should be obtained from the **InitBest** field for the entry corresponding to the current mutation sequence from the **run-Name** file specified in **MutSearch.cfg** (see Sec. 5.1.2 for details). The **bestScore** parameter is used to make sure that the partition function ensemble generated by **doSinglePartFn** will be the same as the corresponding partition function ensemble generated by **KSMaster**.

protPartFn

This parameter is only taken into account for the bound partition function computation (i.e., when **ligPresent** is *true*). The value of this parameter should be obtained from the **Partial_q_E** field for the entry corresponding to the current mutation sequence from the **runName** file specified in **MutSearch.cfg** (see Sec. 5.1.2 for details). The **protPartFn** parameter is used to make sure that the partition function ensemble generated by **doSinglePartFn** will be the same as the corresponding partition function ensemble generated by **KSMaster**.

5.2.2 Output Files

Pairwise Energy Matrices

The **minEnergyMatrixName**/**maxEnergyMatrixName** matrices generated by **KSMaster** are as described for the **doDEE** command (see Sec. 5.1.2). As long as the mutation search parameters remain invariant, the matrices computed by **doDEE** may be used in the **KSMaster** computation, and vice versa. In addition to the **minEnergyMatrixName**/**maxEnergyMatrixName** matrices, if the **useUnboundStruct** parameter in **MutSearch.cfg** is set to *true*, an additional pair of matrices is generated, as specified by the **minEnergyMatrixNameUnbound** and **maxEnergyMatrixNameUnbound** parameters. This pair of matrices have the same format as the corresponding **minEnergyMatrixName**/**maxEnergyMatrixName** matrices. The difference is that **minEnergyMatrixNameUnbound**/**maxEnergyMatrixNameUnbound** are specifically used for the unbound partition function computation in a K^* search, based on the **unboundPdbName** input structure. In that case, **minEnergyMatrixName**/**maxEnergyMatrixName** are only used for the bound partition function computation. If **useUnboundStruct** is *false*, then only the **minEnergyMatrixName**/**maxEnergyMatrixName** matrices are computed and used both for the unbound and bound partition function computation. If **doMinimize** is *false* (i.e., traditional DEE is used with K^*), then only the **minEnergyMatrixName** and **minEnergyMatrixNameUnbound** (if **useUnboundStruct** is *true*) matrices are computed and used.

Mutation Search Results

After the computation of the pairwise energy matrices is done, the program moves to the mutation search. The first stage in the K^* mutation search is the computation of the list of candidate mutation sequences. The list of candidate mutation sequences is stored in the **mutFileName** file specified in **MutSearch.cfg**. If the **mutFileName** file exists in the code directory, that file is read in by K^* and the program continues to the next stage, the K^* score computation. If the **mutFileName** file does not exist, it is computed by K^* .

The first step in this stage is to read in the **volFile** file specified in **KStar.cfg**. This file contains volume information for each rotamer for each amino acid type other than Pro. If this file does not exist, it is computed by K^* . The volume computation uses pre-specified atomic radii and is performed for each rotamer for each amino acid. Each line in the **volFile** file starts with the three-letter code for a given amino acid type i , followed by r_i numbers which correspond to the computed volumes for the r_i rotamers for amino acid i (the rotamers are specified by the respective rotamer library, see Sec. 3.2.1).

Each line in the **mutFileName** file represents a unique candidate mutation sequence. The set of candidate mutation sequences is determined as a function of the following parameters from **MutSearch.cfg**: **numMutations**, **resAllowed**, **addWT**, **targetVolume**, and **volumeWindow**. Let k be the value of **numMutations** and let n be the total number of mutable residue positions. K^* then generates all possible combinations in which the n residue positions are mutated in 1, 2, ..., up to k positions. For each possible combination of mutated/non-mutated positions, all residue combinations are generated using the corresponding position-specific allowed amino acid types specified by the **resAllowed** parameters; if **addWT** is *true*, a position is allowed to also keep its wildtype identity when mutated. For each possible combination of mutated/non-mutated positions, the non-mutated positions are kept in their wildtype identity. This generates a set of candidate mutation sequences. Each unique sequence is then subjected to the volume filter: the sum of the amino acid volumes for the current amino acid assignment to the n positions is checked to determine if that sum falls within the range **targetVolume** \pm **volumeWindow**. If the current sum of volumes is within that range, the corresponding mutation sequence passes the volume filter and is kept for the following K^* computation stage; otherwise, the mutation sequence is considered as over- or under-packed relative to the wildtype sequence/ligand complex, and is thus pruned from further consideration. Each line in the **mutFileName** file corresponds to a sequence that has passed the K^* volume filter. The format of each line is as follows:

c_1 c_2 a_1 a_2 ... a_n

The values of the first two numbers c_1 and c_2 can be ignored (c_2 represents the sum of the volumes for the current amino acid assignment to the n mutable positions; however, this value is not used by K^* after the volume filter checks). The a_i values give the three-letter amino acid codes for the current amino acid assignment to the n mutable positions. An example of a partial **mutFileName** file may look like this:

```
0.0 584.09375 ALA ALA ARG ILE ALA ALA ILE
0.0 583.9844 ALA ALA THR ILE ALA ALA TRP
0.0 581.40625 ALA ALA THR ILE ALA MET ILE
0.0 599.4219 ALA ALA THR ILE ALA PHE ILE
0.0 624.7344 ALA ALA THR ILE ALA TRP ILE
```

NOTE: In some cases, the user may prefer to manually generate the list of candidate mutation sequences to be evaluated by K^* . This can be done by manually generating the **mutFileName** file, as long as the format of each line (two arbitrary numbers followed by the three-letter amino acid codes for the current mutation sequence) is observed.

NOTE: Currently, the volume filter cannot be used for redesigns with ligands that are not natural amino acids. If the ligand is not a natural amino acid, it is recommended to set the **volumeWindow** parameter to a very large value (e.g., 1000000.0), so that no mutation sequences can be pruned by the volume filter.

After the list of candidate mutation sequences is generated and saved, each of the mutation sequences is distributed to a separate processor for evaluation (if the number of sequences exceeds the number of available processors, a queue is formed and sequence distribution continues until the K^* score computation for all sequences completes). Once the K^* computation for a given sequence completes, the results for that sequence are stored in the file specified by the **runName** parameter in **MutSearch.cfg**. The format for each line in that file (corresponding to a single completed

mutation sequence) is the following (line is wrapped):

```

Completed mutation m Score s Volume v SlaveNum -1 Time t1 t2 InitBest s1 FinalBest s2 a1
a2 ... an EConfInfo c1 c2 c3 c4 ELConfInfo c5 c6 c7 c8 MinEMinimized em1 em2 MinEUnMinimized
ei1 ei2 Partial_q_E qp Partial_q_EL qpl E_total cp EL_total cpl ESecondEw b1 ELSecondEw b2
E_allPruned b3 EL_allPruned b4 q_L ql

```

Each line in the **runName** file contains several types of information related to the K^* computation for the corresponding mutation sequence. However, to rank mutation sequences, it is sufficient to only look at the **Score** value s (which gives the computed K^* score for the current mutation sequence) and the a_k values, $1 \leq k \leq n$ (which give the amino acid types for the n mutable residue positions in the current mutation sequence). Once the computation for all mutation sequences is complete, the **runName** file can be read in, and sequences can be sorted according to the **Score** value s . The higher the value s , the better binder the corresponding sequence is predicted to be. The **doSinglePartFn** command can then be run to generate the structures from the K^* ensembles (for the bound or unbound partition function computation) for a user-selected set of the top mutation sequences (see below).

The rest of the data output for each mutation sequence (each line in the **runName** file) is briefly described next.

- m : an index assigned by the program to each mutation sequence. This index does not have any specific meaning and need not be unique (e.g., multiple sequences may have the same index if a mutation search is resumed, see below).
- v : the volume for the current mutation sequence (mutable positions only), computed as described above.
- t_1, t_2 : the time (in minutes) for computing the partition functions for, respectively, the unbound and bound states.
- s_1 : the best score found in the search at the time when the computation for the current mutation sequence was started. Note that this score is not necessarily the overall best score, since scores are continuously updated as more sequences complete their computation.
- s_2 : the best score found in the search at the point where the computation for the current mutation sequence completes.
- c_1, c_2, c_3, c_4 : the number of conformations for the unbound partition function computation that are, respectively, enumerated by K^* , pruned in the DEE pruning stage, pruned by the steric filter, and pruned by the A^* filter (for details of the different filters, see [10]).
- c_5, c_6, c_7, c_8 : the number of conformations for the bound partition function computation that are, respectively, enumerated by K^* , pruned in the DEE pruning stage, pruned by the steric filter, and pruned by the A^* filter.
- e_{m_1}, e_{m_2} : the lowest energy after minimization for any conformation in the, respectively, unbound and bound partition function computation.
- e_{i_1}, e_{i_2} : the lowest energy before minimization for any conformation in the, respectively, unbound and bound partition function computation.
- q_p : the value of the computed unbound partition function for the current mutation sequence.

- q_{pl} : the value of the computed bound partition function for the current mutation sequence with the ligand.
- c_p : the total number of conformations for the unbound partition function computation ($c_p = c_1 + c_2 + c_3 + c_4$).
- c_{pl} : the total number of conformations for the bound partition function computation ($c_{pl} = c_5 + c_6 + c_7 + c_8$).
- b_1 : a boolean value (*true* or *false*) that determines whether the partition function computation for the unbound state was repeated (see the **repeatSearch** parameter in **MutSearch.cfg**).
- b_2 : a boolean value (*true* or *false*) that determines whether the partition function computation for the bound state was repeated.
- b_3 : a boolean value (*true* or *false*) that determines whether all conformations for the current mutation sequence were pruned in the unbound partition function computation. This can happen if, e.g., there are no sterically-allowed conformations for the current mutation sequence.
- b_4 : a boolean value (*true* or *false*) that determines whether all conformations for the current mutation sequence were pruned in the bound partition function computation.
- q_l : the value of the computed unbound partition function for the ligand.

Resuming a mutation search. In some cases (e.g., due to unexpected processor restarts), the mutation search may be interrupted before the computation for all sequences completes. An interrupted mutation search can be resumed, so that only mutation sequences for which the computation did not complete are re-distributed for evaluation. To resume an incomplete search, the **runName** file must be renamed to the filename specified by the **resumeFileName** parameter in **MutSearch.cfg** and the **resumeSearch** parameter should be set to *true*. The completed results from the partial **runName** file (which is now the **resumeFileName** file) are then read in, and only the mutation sequences that are not yet computed are distributed for evaluation. The resumed search generates a new **runName** file with the newly-computed sequence results. If a search must be resumed more than once, all of the previously-completed sequence results must be concatenated into the **resumeFileName** file. When the computation for all sequences completes, the **resumeFileName** and **runName** file must be concatenated to generate the complete sequence results file.

Structure Generation

The **doSinglePartFn** command can be executed separately for a number of user-selected mutation sequences in order to generate pdb structures for the respective K^* ensembles from the bound or unbound partition function computation. Each structure from the respective ensemble is output to a file with the following filename: the concatenation of (1) the **pdbPrefix** parameter in **MutSearch.cfg**, and (2) the conformation number for the given structure, as determined by the order in which the structures are generated (i.e., the first structure from the ensemble will have a conformation number of 1, the next one will have a conformation number of 2, etc.). **NOTE:** The user must make sure that either the **pdbPrefix** parameter has a different value for each different mutation sequence evaluated by **doSinglePartFn**, or that all pdb structures are saved in a unique folder, to guarantee that generated structures are not overwritten upon new calls to **doSinglePartFn**.

5.3 Residue Entropy Computation

OSPREY can apply a Self-Consistent Mean Field (SCMF) approach to compute residue entropies for each residue position in a protein. By using SCMF, self- and pairwise rotamer energies can be computed and used to compute rotamer, and subsequently, amino acid probabilities for any given position in a protein [19]. Residue positions with high entropy can then be redesigned using MinDEE/ A^* to predict mutations anywhere in a protein. This hybrid SCMF and MinDEE/ A^* approach was used in [1] to predict bolstering mutations both close to and far from an enzyme's active site for additional improvement in the enzyme specificity (active site mutations were first identified using the K^* algorithm).

The SCMF computation for a given protein can be performed by OSPREY using the following command:

```
doResEntropy System.cfg ResEntropy.cfg
```

The **System.cfg** and **ResEntropy.cfg** configuration files are described in detail below (Sec. 5.3.1). The output of the **doResEntropy** command is described in detail in Sec. 5.3.2.

NOTE: The residue entropy computation is only performed for a protein. If the input structure contains a ligand, the ligand is deleted from the structure before the residue entropy computation.

5.3.1 Configuration Files

System.cfg

A typical file will look like this:

```
pdbName 1amuH.pdb
pdbLigNum 550
ligAA true
numCofRes 1
cofMap 551
```

All of the parameters are as described for the **System.cfg** file in Sec. 5.1.1. Note that the **System.cfg** file here does not include the two parameters **numInAS** and **residueMap**.

ResEntropy.cfg

This configuration file contains the information about the SCMF residue entropy computation. A typical file will look like this:

```
stericE 30.0
maxPairE 1000.0
useEref true
dist 8.0
matrixName 1amu_resBBPEM
rotProbFile 1amu_rotProb
runName 1amu_resEntropy
```

A description of the parameters is as follows:

stericE

Rotamers with intra-rotamer plus rotamer-to-template energy greater than this threshold are pruned as a preprocessing step.

maxPairE

Rotamer pairwise energies greater than maxPairE are reset to maxPairE.

useEref

Determines if amino acid reference energies are used as part of the energy function.

dist

Distance cutoff for considering two residues as interacting for the energy computation. This parameter is also used as a cutoff for computing the number of neighboring (proximate) residues to a given residue position (see [1, Supporting Information, Sec. S1.2.3]).

matrixName

Prefix for the filenames of the different output matrices described in Sec. 5.3.2.

rotProbFile

The file to which the computed rotamer probabilities are output. This file contains the computed rotamer probabilities for each rotamer, for each amino acid type, for each residue position in a protein. The residue entropies are computed using the rotamer probabilities saved in this file. This is a binary file and is described in Sec. 5.3.2.

runName

The file to which the computed residue entropies are output. The format of this file is described in detail in Sec. 5.3.2.

5.3.2 Output Files

Energy and Other Output Matrices

The first step in the residue entropy computation is the computation of the self- and pairwise rotamer energies. Four different matrices are computed and stored in the main program code folder. These matrices have filenames starting with the **matrixName** parameter as a prefix. The matrix ending in ‘**_dist.dat**’ is a boolean matrix that determines if two residues are within the cutoff interaction distance specified by the **dist** parameter. The matrices ending in ‘**_intra**’ and ‘**_shll**’ contain the intra-rotamer and rotamer-template energies, respectively, for all rotamers for each amino acid type at each residue position. The file ending in ‘**_pair**’ contains a pointer to all computed rotamer-rotamer pairwise matrices in the **peme** subfolder; the full rotamer-rotamer pairwise matrix is divided into multiple smaller pairwise matrices as a way to somewhat decrease the memory requirements for very large systems. The file ending in ‘**_pair**’ is the only plain-text **matrixName** file; all other **matrixName** files are binary files.

NOTE: A file with filename specified by the **runName** parameter concatenated with ‘**.log**’ is generated during the matrix computation. This file contains information about the completed

distributed jobs for a given matrix, and is only used to update the user for the current progress of the computation. This file can be deleted at the end of the computation.

NOTE: The SCMF computation typically requires more memory resources (especially for very large systems), so in some cases **java** may need be executed with larger values for **-Xmx**, e.g., **-Xmx4096M**.

NOTE: The user must make sure that all files generated by a given SCMF run (including the files in the **peme** subfolder) are moved and saved to another location, in order to avoid the possibility of overwriting or unintentionally using existing files for subsequent SCMF runs on different systems.

NOTE: Once any of the matrices described here is computed and saved, it can be reloaded and will not require re-computation if the program execution is interrupted and needs to be restarted. Unlike the K^* and GMEC-based redesigns, no specific flag needs to be set for the program execution to resume. When resumed, the program automatically reads in all computed matrices.

Residue Entropy Results

Once all of the energy matrices have been computed and stored, the program moves to the actual residue entropy computation. At this point in the computation, the program requires only a single processor for execution. First, the probabilities for each rotamer for each amino acid type (other than Pro) for each non-Pro residue position in the protein are computed using SCMF. As a pre-processing step, rotamers with intra-rotamer plus rotamer-to-template energy greater than the value of the **stericE** parameter are pruned from further consideration. Pairwise rotamer energies greater than the value of the **maxPairE** parameter are reset to **maxPairE**. The SCMF rotamer probability computation then continues using a temperature annealing scheme that starts at 50000 and ends at 300 in steps of 100.

Once the SCMF computation is done, the computed rotamer probabilities are saved to the file specified by the **rotProbFile** parameter in **ResEntropy.cfg**. This is a binary file that can also be read in (once computed) if the computation is resumed. Based on the computed rotamer probabilities, the program then computes the corresponding amino acid probabilities for each non-Pro residue position in the input protein. Finally, using the computed amino acid probabilities, the residue entropy of each non-Pro residue position in the protein is computed. The computed residue entropies, as well as the amino acid probabilities for each residue position, are output to the **runName** file. The format of the **runName** file is described next.

The first line in the **runName** file is a header line that contains the column headers for the remainder of the file and has the following format:

```
resNum pdbResNum resDefault entropy  $a_1$   $a_2$  ...  $a_m$  numProx
```

Here, a_i is the three-letter code for amino acid type i (for a total of m amino acid types). The order in which the amino acid types are found in the header line depends on the order in which the amino acid types are read in from the **rotFile** rotamer library (see Sec. 3.2.1). If the default rotamer library is used, the header line in the **runName** file will look like this:

```
resNum pdbResNum resDefault entropy ALA VAL LEU ILE PHE TYR TRP CYS MET  
SER THR LYS ARG HIP ASP GLU ASN GLN GLY numProx
```

The remainder of the **runName** file is organized as follows. Each line in the file corresponds

to a residue position in the protein and has the following format:

r_1 r_2 d e p_1 p_2 ... p_m c

The data values are described as follows:

- r_1 : a residue index for the current residue position. This is typically different from the corresponding residue number from the input pdb structure (see the description of r_2).
- r_2 : the corresponding residue number as read in from the input pdb structure.
- d : the three-letter code of the wildtype amino acid identity for the current residue position.
- e : the computed residue entropy for the current residue position.
- p_1, \dots, p_m : for the current residue position, the computed amino acid probabilities for amino acid types a_1, \dots, a_m .
- c : the number of neighboring (proximate) residue positions as determined by using the **dist** parameter from **ResEntropy.cfg**.

An example line in the **runName** file may look like this (line is wrapped):

```
144 161 ILE 1.287177995617508 0.00929384361661082 0.1865197940818237 0.0018160912871596096
0.6212177621704273 5.759059371666681E-6 0.0 0.025903650444377087 0.044798603334293544
0.06779569113561482 0.006199166859597977 0.01621958547873389 6.625486153748881E-4
3.2651327046301967E-4 7.612305618644515E-4 9.587107682972999E-5 7.080503569800633E-4
0.0033621847202647053 0.008280530237428764 0.00603312369278387 55
```

Residue positions can then be ordered according to their e values (the computed residue entropies). Typically, residue positions with high e values are selected for a subsequent MinDEE/ A^* mutation search. Additionally, the number c of neighboring (proximate) residues can be used to filter out residue positions with too few neighboring residues (see [1, Supporting Information, S1.2.3] for details). The MinDEE/ A^* mutation search can also be limited to include only amino acid types with high probabilities (the p_i values) for a selected residue position.

Chapter 6

Special Types of Redesign

OSPREY is optimized for redesigning proteins and for designing protein-small molecule interactions. It is possible, however, to perform redesigns for other types of problems. This chapter provides instructions for applying OSPREY to two special cases of redesign problems: modeling protein-protein and protein-peptide interactions (Sec. 6.1) and modeling explicit water molecules (Sec. 6.2).

6.1 Protein-Protein and Protein-Peptide Redesign

In the current OSPREY version, protein-protein and protein-peptide interactions can be modeled with certain limitations. Peptides can be modeled as long as the peptide consists only of natural amino acids. A newer version of OSPREY that incorporates many improvements for performing protein-protein and protein-peptide redesign is currently being developed and may be available in the future.

Since OSPREY does not require that the protein structure include sequential residues (i.e., there can be gaps with respect to the protein sequence), two proteins or a protein/peptide complex can be included as part of the input structure and redesigned. In that case, OSPREY will not model the second protein or the peptide as a ligand. Instead, all protein and peptide residues will be modeled as one (virtual) protein, in which selected residues can be allowed to mutate, change their side-chain conformation, and energy-minimize.

As an example, let us consider a protein-protein complex (consisting of proteins *A* and *B*) in which both proteins have a total of 150 residues. Let the input pdb structure **prot.pdb** be formatted according to the instructions in Sec. 4.1. Specifically, as discussed in Sec. 4.1, OSPREY has no notion of chain IDs and no two residues in the input structure should have the same residue numbers. Hence, let the residues for protein *A* have residue numbers between 1 and 150, and the residues for protein *B* - between 151 and 300. The selected flexible/mutable residue positions in both proteins must be included in the **residueMap** parameter in **System.cfg** (see Sec. 5.1.1). Let the flexible/mutable residue positions in *A* be 5, 7, 20, and in *B* - 220, 230, 232, and 233. The **System.cfg** may then look like this:

```
pdbName prot.pdb
numInAS 7
residueMap 5 7 20 220 230 232 233
```

```
pdbLigNum -1
ligAA false
numCofRes 0
cofMap -1
```

NOTE: Currently, ligands in OSPREY can only be a single amino acid or small molecule. This is the reason for setting **pdbLigNum** to **-1** (which effectively means there is no ligand in the input structure). Instead, both proteins are treated as part of one (virtual) protein molecule. Side-chain and backbone flexibility can nevertheless be modeled for both proteins. However, no rigid-body rotation and translation will be allowed under this setup (rotations/translations are only allowed for the ligand).

NOTE: When modeling protein-peptide interactions, the current backbone flexibility approaches in OSPREY may not be sufficient for the inherently more flexible peptides.

NOTE: Since the second protein cannot be modeled as a ligand, the K^* algorithm cannot be applied. Only the GMEC-based DEE/ A^* algorithms can be applied to protein-protein and protein-peptide redesign problems.

NOTE: A newer version of OSPREY that incorporates many improvements for performing protein-protein and protein-peptide redesign is currently being developed and may be available in the future.

6.2 Modeling Explicit Waters

The EEF1 solvation model (Sec. 4.3) is the standard way of modeling solvation energies in OSPREY. In some cases, however, explicit water molecules can also be included as part of the input structure and used in the energy computation for the mutant structures. In the current version of OSPREY, explicit water molecules can be modeled as rigid (virtual) cofactor residues. Several considerations must be taken into account when modeling explicit water molecules:

- Water molecules can only be modeled as rigid: all explicit waters remain rigid in their position from the input structure.
- All explicit waters must also include both hydrogens. The orientation of the hydrogens is also kept rigid and must thus be optimized before starting the OSPREY mutation search.
- The three-letter code for each water molecule (residue) should be **HOH**. Each water molecule must have three atoms with names: **O**, **H1**, and **H2**.
- It is recommended to only include water molecules that participate in important interactions, but that are likely to remain in their input position/conformation and not be displaced by mutated neighboring residues.

To build on the example **System.cfg** from Sec. 6.1 above, let us also model two water molecules as part of the input structure. Let these two water molecules have residue numbers 310 and 320 (note that the residue numbers must be unique, as discussed in Sec. 4.1) and the following coordinates (pdb format):

ATOM	15580	O	HOH	310	59.519	21.013	23.619	1.00	12.33
ATOM	15581	H1	HOH	310	59.317	21.271	22.674	1.00	19.69
ATOM	15582	H2	HOH	310	60.165	20.253	23.540	1.00	19.69
ATOM	15610	O	HOH	320	54.153	21.474	18.635	1.00	13.78
ATOM	15611	H1	HOH	320	53.271	21.803	18.297	1.00	19.69
ATOM	15612	H2	HOH	320	54.628	21.145	17.819	1.00	19.69

The `System.cfg` from Sec. 6.1 should then be modified in the following way:

```

pdbName prot.pdb
numInAS 7
residueMap 5 7 20 220 230 232 233
pdbLigNum -1
ligAA false
numCofRes 2
cofMap 310 320

```

NOTE: The considerations for the input structure described in Sec. 4.1 must be followed when modeling explicit water molecules.

Chapter 7

OSPREY via an Example

This chapter presents a detailed example of how to apply K^* to redesign an enzyme in order to switch its specificity toward a novel substrate. The goal of this (over-simplified) example is, however, not to present results of biomedical significance; rather, the goal is to assist the user in setting up and applying OSPREY for their own protein design problems. **NOTE:** This example describes *how* protein redesign can be performed using K^* . This example does not explain *why* certain steps are performed. While this example captures a large portion of the K^* specifics, there are a number of special considerations not covered here. Moreover, DEE/ A^* and SCMF-based redesigns are not described here. The user is therefore urged to also read the entire user manual, and especially Chapter 4, before reading through this example.

OSPREY setup

The setup for performing a K^* redesign can be divided into several sequential steps. The details of each step are described below. We will assume that OSPREY and all related software have been setup and are ready for use, as described in Chapter 2. The input files for this example are included in the `example/input` folder in the OSPREY distribution.

1. **Select a redesign system.** We choose to redesign the phenylalanine adenylation domain of the non-ribosomal peptide synthetase enzyme Gramicidin S Synthetase A (GrsA-PheA) for a non-cognate substrate.
2. **Select mutable positions.** We select six of the GrsA-PheA active site residues for mutation: 239, 278, 299, 301, 322, and 330.
3. **Obtain pdb input structure.** The crystal structure of GrsA-PheA in complex with Phe and the cofactor AMP is available (PDB id: 1amu [2]). We download the structure and format it according to the instructions in Sec. 4.1. Specifically, we:
 - *Add hydrogens.* We use MolProbity [4] to protonate the input structure. The resulting file is `1amuFH.pdb` in the `example/input` folder included with the OSPREY distribution. **NOTE:** When adding hydrogens to 1amu, MolProbity could not add H_1 , H_2 , and H_3 to the Phe substrate; we can add these missing hydrogens using another external program (e.g., Accelrys Discovery Studio Visualizer).

- *Select a steric shell.* Since GrsA-PheA has more than 500 residues, we choose to only include a steric shell of residues close to the active site of the enzyme as part of the OSPREY input structure. First, we select the GrsA-PheA complex from Chain A in the crystal structure. We delete all of Chain B and all water molecules; we also delete Mg and SO₄ from Chain A. At this point, the current structure consists of the following from Chain A: all protein residues, the Phe substrate, and the AMP cofactor. The resulting file may look like **1amuFH_A.pdb** in the **example/input** folder. To extract the steric shell from this structure, we can use an external program (e.g., VMD [11]). Typically, the shell is selected so that all residues close to the active site residues are included. For simplicity, in this example, we choose to only include the residues within 8 Å from the Phe substrate (this also includes the AMP cofactor). The file resulting from this step may look like **1amuFH_A_8-lig.pdb** in the **example/input** folder.
 - *Rename HETATM to ATOM.* If there are HETATM lines in the input structure, rename HETATM to ATOM. **NOTE:** Remember to add two extra spaces after the word ATOM in order to observe the standard PDB column widths. In this example, **1amuFH_A_8-lig.pdb** contains no HETATM lines, so this step can be skipped.
 - *Rename HIS residues.* This step is performed according to the instructions in Sec. 4.1. In this example, **1amuFH_A_8-lig.pdb** contains no HIS residues, so this step can be skipped.
 - *Check final structure.* The final structure (**1amuFH_A_8-lig.pdb** in this example) should be checked for missing atoms, protonation states, and other considerations, as described in Sec. 4.1. Specifically, every line should be an ATOM line and the last line should be an END line; all residues must have unique residue numbers (chain ID's are not recognized).
4. **Select a redesign target substrate.** The wildtype substrate of GrsA-PheA is Phe. Our choice is to redesign GrsA-PheA for a small molecule ligand with three-letter name 'FCL' that differs from Phe in that chlorine is added to the para ring position (Fig. 4.1). FCL was originally described in Sec. 4.3.
 5. **Obtain substrate coordinates.** Since the target substrate is not a natural amino acid, it must be part of the input structure: i.e., the Phe substrate in **1amuFH_A_8-lig.pdb** must be replaced by FCL in the OSPREY input structure before performing the K^* redesign. We can use an external program, such as PyMOL [5], to build the FCL molecule from the Phe substrate. We save the resulting structure as **1amuFH_A_8-lig_FCL.pdb** in the **example/input** folder. **NOTE:** OSPREY does not perform docking, so the target substrate must already be docked in the binding site. Changes to the substrate conformation are allowed through rotamer swaps and (for designs using MinDEE) side-chain dihedral minimization, as well as bounded rigid-body rotation and translation.
 6. **Add missing force field parameters.** The force field parameters for FCL are computed as described in the example in Sec. 4.3 and are shown in Fig. 4.2. These parameters are added to the **all_nuc94_and_gr.in** file as described in Sec. 4.3. Additional modifications must be made to the **parm96a.dat** file (see Sec. 4.3 for details). The resulting modified **all_nuc94_and_gr.in** and **parm96a.dat** files can be found in the **example/input** folder.

7. **Obtain FCL rotamers.** For the FCL rotamers, we will use the Phe rotamers from the Penultimate rotamer library [17]. The FCL rotamers must be added to the file specified by the `grotFile` parameter in `KStar.cfg` (Sec. 3.2.1). We will assume this file is `GenericRotamers.dat`. The following must be added to that file, following the instructions in Sec. 4.2 (remember to increase the first non-`!` line from `1` to `2`):

```
FCL 2 4
N CA CB CG
CA CB CG CD1
62 90
-177 80
-65 -85
-65 -30
```

The resulting modified `GenericRotamers.dat` file can be found in the `example/input` folder.

8. **Setup configuration files.** The `KStar.cfg`, `System.cfg`, and `MutSearch.cfg` configuration files that will be used in this example are shown below. A detailed description of these configuration files, as well as some special considerations, is given in Secs. 3.2.1, 5.1.1, and 5.2.1, respectively. We will only note that in this example, K^* will aim at predicting up to 2-point mutation sequences, as determined by the `numMutations` parameter in `MutSearch.cfg`.

KStar.cfg:

```
hElect true
hVDW false
hSteric false
distDepDielect true
dielectConst 6.0
vdwMult 0.95
doDihedE true
doSolvationE true
solvScale 0.8
stericThresh 0.4
softStericThresh 1.5
rotFile LovellRotamer.dat
grotFile GenericRotamers.dat
volFile AAVolumes.dat
```

System.cfg:

```
pdbName 1amuFH_A_8-lig_FCL.pdb
numInAS 6
residueMap 239 278 299 301 322 330
pdbLigNum 566
```

```
ligAA false
numCofRes 1
cofMap 567
```

MutSearch.cfg:

```
runName 1amuFCL_MutSearch
mutFileName 1amuFCL_2MUT.mut
numMutations 2
targetVolume 620.0
volumeWindow 100000000.0
doMinimize true
minimizeBB false
doBackrubs false
backrubFile none
minEnergyMatrixName 1amuFCL_SCPEMmin
maxEnergyMatrixName 1amuFCL_SCPEMmax
initEw 6.0
pruningE 100.0
stericE 30.0
scaleInt false
maxIntScale 1.0
epsilon 0.03
gamma 0.01
repeatSearch true
useUnboundStruct false
unboundPdbName none
minEnergyMatrixNameUnbound none
maxEnergyMatrixNameUnbound none
ligPresent true
ligType fcl
addWT true
resAllowed0 gly ala val leu ile tyr phe trp met
resAllowed1 gly ala val leu ile tyr phe trp met
resAllowed2 gly ala val leu ile tyr phe trp met
resAllowed3 gly ala val leu ile tyr phe trp met
resAllowed4 gly ala val leu ile tyr phe trp met
resAllowed5 gly ala val leu ile tyr phe trp met
resumeSearch false
resumeFileName 1amuFCL_MutSearch.partial
```

Performing the K^* redesign

At this point, we have setup OSPREY for performing a K^* redesign of GrsA-PheA for the target substrate FCL. To perform the K^* redesign, we execute the following OSPREY command from the

shell:

```
mpirun -machinefile ./machines -np 5 java -Xmx1024M KStar mpi -c KStar.cfg KSMaster
System.cfg MutSearch.cfg >! logKS.out
```

In this example, we will be running OSPREY on 5 processors. The output files generated by the K^* computation for this example can be found in the **example/output** folder in the OSPREY distribution. The standard output is redirected to a file called **logKS.out**. Next, we show the output as seen by the user. Output generated by the K^* computation to the **logKS.out** file is shown in **this font**. We will also include comments explaining the **logKS.out** output; these comments are in bold and enclosed in brackets: [**this is a comment**]. The following signifies one or more skipped lines of output: [.....].

K^* execution

[Upon startup, the OSPREY program information is displayed first, as described in Sec. 3.2:]

```
OSPREY Protein Redesign Software Version 1.0
Copyright (C) 2001-2009 Bruce Donald Lab, Duke University
```

```
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.
```

```
There are additional restrictions imposed on the use and distribution
of this open-source code, including: (A) this header must be included
in any modification or extension of the code; (B) you are required to
cite our papers in any publications that use this code. The citation
for the various different modules of our software, together with a
complete list of requirements and restrictions are found in the
document license.pdf enclosed with this distribution.
```

```
OSPREY running on 5 processor(s)
```

[Next, the program outputs some of the parameter values that were read from the input configuration files:]

```
Run Name: 1amuFCL_MutSearch
Precomputed Min Energy Matrix: 1amuFCL_SCPEMmin
Precomputed Max Energy Matrix: 1amuFCL_SCPEMmax
Ligand Type: fcl
Volume Center: 620.0
Volume Window Size: 1.0E8
Num Residues Allowed to Mutate: 2
```

ResidueMap: 16(TRP A 239) 19(THR A 278) 23(ILE A 299) 25(ALA A 301) 30(ALA A 322) 38(ILE A 330)

[The program then checks if the pairwise energy matrices have been computed. If yes, the matrices are loaded and the program execution continues to the mutation search; if no, then the matrices are automatically computed:]

Checking if precomputed energy matrix is already computed...Precomputed energy matrices not available..

Run Name: 1amuFCL_MutSearch

Precomputed Minimum Energy Matrix: 1amuFCL_SCPEMmin

Precomputed Maximum Energy Matrix: 1amuFCL_SCPEMmax

Ligand Type: fcl

Num Residues Allowed to Mutate: 2

Computing _All_ Rotamer-Rotamer Energies

Starting minimum and maximum bound energy computation

ResidueMap: 16(TRP A 239) 19(THR A 278) 23(ILE A 299) 25(ALA A 301) 30(ALA A 322) 38(ILE A 330)

Beginning setAllowables

Number of possible mutation combinations: 15

Length of mutArray: 29

Retrieving 0 of 29

Sent to proc 1

Retrieving 1 of 29

Sent to proc 2

Retrieving 2 of 29

Sent to proc 3

Retrieving 3 of 29

Sent to proc 4

Beginning setAllowables

Beginning setAllowables

Beginning setAllowables

Beginning setAllowables

.....Finished: 3, Time: 2.7333333333333334

Retrieving 4 of 29, Sent to proc 4

[.....]

[The pairwise matrix computation is divided into several parts, and each part is distributed for evaluation by a separate processor. In this example, there are a total of 29 matrix parts. We are running on a total of 5 processors, so 1 processor serves as the main compute node and the remaining 4 processors serve as work nodes. Hence, the computation for four matrix parts is distributed to the work nodes. Once the computation for one of these parts completes (part 3 here), the next matrix part is retrieved and submitted to the corresponding work node. The distributed matrix computation continues until all of the 29 jobs complete:]

.....Finished: 28, Time: 4.45

```
DONE: Pairwise energy matrix precomputation..
PEM execution time: 18.249733333333333
DONE: Pairwise energy matrix precomputation
done
```

[Note that each matrix part may require a different amount of time to complete, so a job that was distributed at the beginning of the computation may in fact complete last, and vice versa. Once the pairwise energy matrix computation is done, the two (when energy minimization is performed) or one (with no energy minimization) energy matrices are output. In this example, the filenames of the two energy matrices are: 1amuFCL_SCPEMmin.dat and 1amuFCL_SCPEMmax.dat.

Next, the program looks for the file that contains the list of candidate mutation sequences (see the discussion in Sec. 5.2.2 of the mutFileName file specified in MutSearch.cfg). If the mutFileName file exists, it is read in and the program execution continues to the K^* score computation for the candidate mutation sequences. If that file does not exist, then it is computed by the program, as described in Sec. 5.2.2:]

```
Looking for mutation list file ... no mutation list file found. Computing one.
Starting mutation combination 0 ...
.....
.finished
Starting mutation combination 1 ...
.....
.finished
Starting mutation combination 2 ...
.....
.finished
Starting mutation combination 3 ...
.....
.finished
Starting mutation combination 4 ...
.....
.finished
Starting mutation combination 5 ...
.....
.finished
Starting mutation combination 6 ...
.....
.finished
Starting mutation combination 7 ...
.....
.finished
Starting mutation combination 8 ...
.....
.finished
Starting mutation combination 9 ...
.....
.finished
Starting mutation combination 10 ...
.....
```

```

.finished
Starting mutation combination 11 ...
.....
.finished
Starting mutation combination 12 ...
.....
.finished
Starting mutation combination 13 ...
.....
.finished
Starting mutation combination 14 ...
.....
.....finished
Starting mutation combination 15 ... .....finished
Starting mutation combination 16 ... .....finished
Starting mutation combination 17 ... .....finished
Starting mutation combination 18 ... .....finished
Starting mutation combination 19 ... .....finished
Starting mutation combination 20 ... .....finished
Starting mutation combination 21 ... .finished
Sequences remaining after volume filter 1316
Allocated newArray
Initial Length of mutArray: 200000
Trimmed Length of mutArray: 1316
Removing duplicates...done
1050 unique mutation sequences found in volume range -9.9999376E7 to
1.00000624E8
Total number of conformations (bound and unbound) for all sequences: 26456220
Ligand partition function (double): 0.00356708

```

[As a result of the mutation list computation, we get 1050 candidate mutation sequences. The list of candidate sequences is saved to the file 1amuFCL_2MUT.mut. Note that in this example the volume filter is actually turned off, so no sequence pruning is performed based on volumes. The last two lines from the output above specify the total number of rotamer-based conformations for the bound and unbound states for all candidate mutation sequences and the value of the computed ligand partition function.

Next, the computation of the K^* score for each of the 1050 candidate mutation sequences is distributed to a separate work node for evaluation. Again, the first 4 sequences (equal to the number of available work nodes) are distributed, and once the computation for one of these sequences is complete, a new candidate sequence is retrieved and sent for evaluation.]

```

Retrieving 0 of 1050
Sent to proc 1

Retrieving 1 of 1050
Sent to proc 2

Retrieving 2 of 1050

```


Sent to proc 3

Retrieving 3 of 1050

Sent to proc 4

```
Loading precomputed min energy matrix...Loading precomputed min energy
matrix...Loading precomputed min energy matrix...done
MinDEE: Loading precomputed max energy matrix...Loading precomputed min energy
matrix...done
MinDEE: Loading precomputed max energy matrix...done
Beginning setAllowables
Number of rotamers pruned due to incompatibility with the template: 1
Computing MinDEE interval terms..done.
```

```
ind: 54.678131103515625 54.678131103515625 50.374237060546875
54.678131103515625 54.678131103515625 4.30389404296875
pair: 5.016383171081543 5.694559335708618 5.664247035980225 10.66715407371521
10.666823148727417 5.01689887046814
Current run: 1
Starting AS residue 0.done
Starting AS residue 1.done
Starting AS residue 2.done
Starting AS residue 3.done
Starting AS residue 4.done
[.....]
```

[The K^* score computation for each candidate sequence generates various types of output to the logKS.out file, including DEE pruning information and information related to the ensemble of conformations generated by A^* . Since multiple sequences are evaluated at the same time, the output for the different sequences is intermixed. This output is useful in some special cases but can typically be discarded. When the K^* computation for a given sequence completes, the results for this sequence are output to the file specified by the runName parameter in MutSearch.cfg, as described in Sec. 5.2.2. In this example, this file is 1amuFCL_MutSearch. When the K^* computation for all sequences completes, the following is output:]

```
DONE: K* computation
```

[To verify that the computation for all sequences has indeed completed, we can compare the number of lines in the 1amuFCL_2MUT.mut and 1amuFCL_MutSearch files - these numbers should be equal. In this example, the number of lines in 1amuFCL_MutSearch must be 1050 - we can verify this using, e.g., wc. If the K^* computation is interrupted, we can resume the search using the partial results from 1amuFCL_MutSearch, as described in Sec. 5.2.2.]

Analysis of results

At this point, we have computed K^* scores for all candidate mutation sequences. The sequences can now be ranked in order of decreasing K^* scores, as described in Sec. 5.2.2. The top ten mutants,

Rank	K^* Score	239	278	299	301	322	330
1	$4.25E + 24$	ILE	TRP	ILE	ALA	ALA	ILE
2	$3.12E + 24$	VAL	TRP	ILE	ALA	ALA	ILE
3	$2.18E + 24$	ILE	THR	ILE	PHE	ALA	ILE
4	$1.45E + 24$	VAL	THR	ILE	PHE	ALA	ILE
5	$1.41E + 24$	ILE	THR	ILE	TYR	ALA	ILE
6	$1.40E + 24$	ILE	PHE	ILE	ALA	ALA	ILE
7	$9.55E + 23$	VAL	PHE	ILE	ALA	ALA	ILE
8	$9.27E + 23$	VAL	THR	ILE	TYR	ALA	ILE
9	$8.85E + 23$	ALA	TRP	ILE	ALA	ALA	ILE
10	$6.57E + 23$	MET	TRP	ILE	ALA	ALA	ILE

Table 7.1: Top ten mutants predicted by K^* for the example redesign of GrsA-PheA for FCL. For each of the mutants, the predicted amino acid identities at the six mutable residue positions are shown along with the corresponding K^* rank and computed score.

along with their K^* scores for this example are shown in Table 7.1. **NOTE:** As was already noted, the goal of the (oversimplified) example described in this chapter and the computational predictions is to assist the user in setting up and applying OSPREY for their own system, rather than to present results of biomedical significance.

Visualization of predicted structures

At this point, we have the top mutant predictions as determined by K^* . We can now generate and visualize predicted structures from the K^* ensembles for a selected set of the top mutants. We choose to visualize structures for the bound protein-substrate ensemble of the top-ranked mutant W239I/T278W. To do this, we can use the `doSinglePartFn` command:

```
java -Xmx1024M KStar -c KStar.cfg doSinglePartFn System.cfg SinglePF.cfg >! logSPF.out
```

Note that the `doSinglePartFn` command, unlike `KSMaster`, does not require MPI and distributed execution. Hence, we can simply use the command as shown above, without using `mpirun`. The input/output files as well as the computed structures for this example are included in the `example/struct` folder in the OSPREY distribution. The parameter values of the `SinglePF.cfg` configuration file are given below; this file is described in detail in Sec. 5.2.1.

`SinglePF.cfg`:

```
doMinimize true
minimizeBB false
doBackrubs false
backrubFile none
minEnergyMatrixName 1amuFCL_SCPEMmin
maxEnergyMatrixName 1amuFCL_SCPEMmax
initEw 6.0
pruningE 100.0
stericE 30.0
```

```

scaleInt false
maxIntScale 1.0
epsilon 0.03
gamma 0.01
repeatSearch true
resMut ILE TRP ILE ALA ALA ILE
ligPresent true
ligType fcl
outputPDBs true
pdbPrefix pdbs/1amuFCL_
bestScore 2183091932959800697659312.82422150
protPartFn 159304123380538183833360725603391236707263316455397704887424340607652
35993841335877708506511476171621822096215867891565727481052525690256347324650653
15902489129779248569737575226250840643408253299348510492212113054580162717691191
16957445360544516193553071079829135500741194500513652097139354693011546300590300
49011209136620375459455218984833362811641976000247270723349681855843.96015854

```

The standard output from the **doSinglePartFn** run is redirected to the **logSPF.out** file. Output generated by the K^* computation to the **logSPF.out** file is shown in this font. We will also include comments explaining the **logSPF.out** output; these comments are in bold and enclosed in brackets: **[this is a comment]**. The following signifies one or more skipped lines of output: **[.....]**.

[As usual, the OSPREY program information is displayed first, as described in Sec. 3.2:]

```

OSPREY Protein Redesign Software Version 1.0
Copyright (C) 2001-2009 Bruce Donald Lab, Duke University

```

```

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version.

```

```

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

```

```

There are additional restrictions imposed on the use and distribution
of this open-source code, including: (A) this header must be included
in any modification or extension of the code; (B) you are required to
cite our papers in any publications that use this code. The citation
for the various different modules of our software, together with a
complete list of requirements and restrictions are found in the
document license.pdf enclosed with this distribution.

```

```

OSPREY running on 1 processor(s)

```

[Next, the program outputs some of the parameter values that were read from the input configuration files, including the particular mutation sequence for which we are computing the ensemble structures:]

ResidueMap: 16(TRP A 239) 19(THR A 278) 23(ILE A 299) 25(ALA A 301) 30(ALA A 322) 38(ILE A 330)
Mutation Sequence: ILE TRP ILE ALA ALA ILE
Beginning setAllowables
Loading precomputed min energy matrix...done
Ligand partition function (double): 0.00356708
Before start

[Next, the program performs MinDEE-based pruning, and displays some related information and statistics:]

Number of rotamers pruned due to incompatibility with the template: 11
Computing MinDEE interval terms..done.

ind: 92.58587646484375 148.0013427734375 180.3873291015625 184.69122314453125
184.69122314453125 134.31698608398438 183.47335815429688
pair: 1038.102953672409 565.7130352258682 882.9790832996368 1424.3734160661697
1577.3457723855972 1085.528309226036 1399.7722948789597

Current run: 1

Starting AS residue 0.done
Starting AS residue 1.done
Starting AS residue 2.done
Starting AS residue 3.done
Starting AS residue 4.done
Starting AS residue 5.done
Starting ligand run..done

Number of rotamers pruned this run: 1

DEE: The minimum difference is -844.6013952493668

Current run: 2

Starting AS residue 0.done
Starting AS residue 1.done
Starting AS residue 2.done
Starting AS residue 3.done
Starting AS residue 4.done
Starting AS residue 5.done
Starting ligand run..done

Number of rotamers pruned this run: 0

DEE: The minimum difference is -844.6013952493668

Current run: 1

Starting AS residue 0.done
Starting AS residue 1.done
Starting AS residue 2.done
Starting AS residue 3.done
Starting AS residue 4.done
Starting AS residue 5.done
Starting ligand run..done

Number of rotamers for the current sequence: 34

Number of rotamers pruned this run: 2

minE: 9.999999680285692E37 maxE: 9.999999680285692E37 pruningE: 100.0 Ew: 6.0

Ec: 9.999999680285692E37
Num rotamers pruned due to unallowed sterics (from Bounds): 2

Current run: 2
Starting AS residue 0.done
Starting AS residue 1.done
Starting AS residue 2.done
Starting AS residue 3.done
Starting AS residue 4.done
Starting AS residue 5.done
Starting ligand run..done
Number of rotamers for the current sequence: 34
Number of rotamers pruned this run: 0

minE: 9.999999680285692E37 maxE: -9.999999680285692E37 pruningE: 100.0 Ew: 6.0
Ec: 9.999999680285692E37
Num rotamers pruned due to unallowed sterics (from Bounds): 0

Current run: 1
Starting AS residue 0.done
Starting AS residue 1.done
Starting AS residue 2.done
Starting AS residue 3.done
Starting AS residue 4.done
Starting AS residue 5.done
Starting ligand run..done
Number of rotamers for the current sequence: 34

minE: 9.999999680285692E37 maxE: -9.999999680285692E37 pruningE: 100.0 Ew: 6.0
Ec: -526.9367780685425
Num rotamers pruned due to unallowed sterics (from all criteria): 9

One Mutation Conf Found and is being tested
k_const: 648 pStar:
49067609227542163291289254320528748942616920281842277242212836853134934304413858
05169442422913289354482210093141871661925246446936371645296716819728578241329567
72005815081861329712020364309248952336560041150778092300482899345480040837954190
88077812737098532113522365407343473185245996911800307801115489168951984842807475
329224903146052788956084782164345051973040456654030439645995548964604.43688016
numConfsPrunedMinDEESteric: 8596

[At this point, the A^* search is started and rotamer-based conformations and structures are generated, until the computed ensemble partition function is guaranteed to be within ε of the full partition function, or until the inter-mutation filter determines that the current mutation sequence is not a feasible candidate [10]. Information for each A^* -generated conformation is displayed:]

6 3 5 0 0 4 3
conf: 1 minELowerBound: -548.2034912109375 curThreshold: -535.2999449749552
pStar(double): Infinity qStar(double): 0.0 rho*qStar(double): 0.0
energy: -543.8115

```

6 3 5 0 0 4 2
conf: 2 minELowerBound: -547.7808837890625 curThreshold: -538.2642303245228
pStar(double): Infinity qStar(double): Infinity rho*qStar(double): Infinity
energy: -538.9178
6 3 4 0 0 4 3
conf: 3 minELowerBound: -547.24658203125 curThreshold: -538.2693896359148
pStar(double): Infinity qStar(double): Infinity rho*qStar(double): Infinity
energy: -543.22473
[....]
5 3 2 0 0 4 2
conf: 37 minELowerBound: -539.6468505859375 curThreshold: -538.5397930967628
pStar(double): Infinity qStar(double): Infinity rho*qStar(double): Infinity
energy: -531.16187
5 3 6 0 0 5 2
conf: 38 minELowerBound: -539.3419799804688 curThreshold: -538.5416848056631
pStar(double): Infinity qStar(double): Infinity rho*qStar(double): Infinity
energy: -529.3732
4 3 3 0 0 4 3
conf: 39 minELowerBound: -539.037353515625 curThreshold: -538.5435825678219
pStar(double): Infinity qStar(double): Infinity rho*qStar(double): Infinity
energy: -521.02075
4 3 3 0 0 4 2
conf: 40 minELowerBound: -538.628662109375 curThreshold: -538.545486422314
pStar(double): Infinity qStar(double): Infinity rho*qStar(double): Infinity
energy: -516.04034
4 3 2 0 0 4 3
conf: 41 minELowerBound: -538.54248046875 curThreshold: -538.5473964083933
pStar(double): Infinity qStar(double): Infinity rho*qStar(double): Infinity

```

[Zooming in, four lines of output are generated for each conformation. For example, the output associated with the first conformation shown above is the following:

```

6 3 5 0 0 4 3
conf: 1 minELowerBound: -548.2034912109375 curThreshold: -535.2999449749552
pStar(double): Infinity qStar(double): 0.0 rho*qStar(double): 0.0
energy: -543.8115

```

The first line gives the corresponding rotamer index for each of the mutable residue positions (in this case, six). The second line shows: (1) The number of the current conformation in terms of the A^* generation order; (2) A lower bound on the energy of the conformation (as computed using the precomputed pairwise energy matrices); and (3) The current stop-threshold, as given by [10, Eq. A20]. The A^* enumeration stops when the lower bound on the energy of the current conformation becomes greater than the current stop threshold. The third line shows the current (partial) partition functions; for reasons of space, these values are displayed only with double precision, so larger values are shown as *Infinity*. The last line shows the (minimized) energy computed for the current conformation.

Once the stop threshold is reached, the A^* search halts, and the computation statistics are displayed:]

```

Statistics (bound):

```

```

Best Energy:  -543.8115234375
partial_q:
24141441511867708183603979839649623057702138766052145837054341072536726565446708
91426359712060156094054039210361767126205754941656205661671297671138129480138430
07688833854783673688788184519852040201461963430592031012180996509148653698461346
28683834766342525554361634707288259171995491305342574507130543987377445660250204
3108449723888961228514738993615176234722869747397686011821768244809688536642467.
31489331
partial_p: 0
NumConfsTotal:      9604
NumConfsPrunedByMinDEE: 9244
NumConfsPrunedByS:  10
NumConfsEvaluated:  40
NumConfsLeft:       310

```

For each conformation generated by A^* , the computed structures are also output in pdb format to the folder (and using the filenames) specified by the **pdbPrefix** parameter in **SinglePF.cfg**. In this example, the program has generated forty structures in a subfolder called **pdb**s (the user must make sure this folder exists before the execution of **doSinglePartFn**), with filenames **1amuFCL_1.pdb** through **1amuFCL_40.pdb**. These structures can now be analyzed and visualized using tools like MolProbity [4] and PyMOL [5]. These structures are for the bound state ensemble of the selected mutant. Additionally, structures from the free protein ensemble can also be generated for that mutant by setting the **ligPresent** parameter in **SinglePF.cfg** to *false*. Structures for other mutants can also be generated by running the **doSinglePartFn** command separately for each selected mutant.

With this, we conclude the example of how to perform K^* redesign for a given system. This example covers some of the specifics of using the OSPREY software. However, the user is urged to read the entire user manual since the other OSPREY commands, as well as some other details not relevant to the example described in this chapter, are not described here. Further details on the algorithms used in OSPREY can be found in our publications [1, 10, 7, 8, 9].

Appendix A

OSPREY Class Summary

This Appendix is intended to serve as a starting point for users interested in modifying and extending the OSPREY source code. A summary of the OSPREY classes is as follows:

- **Amber96ext:** This class handles the energy computation for a given molecule. The Amber force field parameters are read in and saved upon initialization. The EEF1 solvation parameters are also read in using the EEF1 class functions. The energy (including electrostatic, vdW, and solvation) and gradient computation for the full molecule or a selected subset of the molecule atoms is performed by this class.
- **Amber96PolyPeptideResidue:** This class contains hard-coded templates for the different amino acid types. These templates are used when performing residue mutations.
- **AminoAcidTemplates:** This class reads from three data files specifying amino acid residues, N-terminal amino acid residues, and C-terminal amino acid residues. Information read includes element type, atom type, limited connectivity, and partial charge. By matching these amino acid templates to actual residues in a molecule, the corresponding template atom types and partial charges can be assigned to the matched residues.
- **Atom:** Handles functions and data associated with atoms. Example functions include adding a bond between atoms, computing torsion, computing atom distance, etc. Some of the data members include the atom name, radius, mass, coordinates, and bond information.
- **Backbone:** Handles the backbone representation for the protein; Applies (ϕ , ψ) changes to the molecule; Assumes that the order of the atoms for the ϕ angle is C(i-1), N(i), CA(i), C(i), and for the ψ angle: N(i), CA(i), C(i), N(i+1).
- **BackrubMinimizer:** Handles two types of energy minimization: (1) the minimization required for computing the pairwise energy matrices; (2) the minimization of a full conformation: the side-chain dihedrals are kept rigid, while the backbone is allowed to move using backrub motions. Currently, Backrub minimization can be applied only to the system strand of the molecule; the ligand (if present) is allowed to rotate and translate.
- **Backrubs:** Handles the application of the Richardsons' Backrub motions for a given residue in a protein.

- **BBMinimizer**: Handles two types of backbone energy minimization for BD: (1) the minimization required for computing the pairwise energy matrices; (2) the minimization of a full conformation: the side-chain dihedrals are kept rigid, while the backbone dihedrals are allowed to move within given limits.
- **BoundFlags**: Applies the Bounding Flags pruning criteria: computes a lower bound on the energy of all conformations that contain a given rotamer pair (i_r, j_s) , for each rotamer pair.
- **CommucObj**: The CommucObj class is a data structure used in communication between the master and slave nodes. It is basically just a data container. It allows the master node to specify what type of search the slave should perform and it allows the slave to return the result of the computation to the master.
- **DEEGoldstein**: Performs simple Goldstein DEE rotamer pruning.
- **DEEGoldsteinPairs**: Performs DEE Goldstein pairs rotamer pruning.
- **DEESplit1f**: Performs full split-DEE (conformational splitting) with 1 split position.
- **DEESplit2f**: Performs full split-DEE (conformational splitting) with 2 split positions.
- **EEF1**: Manages the EEF1 solvation parameters; EEF1 handles only natural amino acids, so solvation energies are computed for proteins and ligands (natural amino acids) only, and not cofactors; Some important notes about assumptions of the EEF1 model are given in the comments to `getSolvationParameters()`.
- **ExpansionQueue**: This queue is ordered in terms of increasing $f(n)$ values of the nodes in the A^* expansion tree; only the visible nodes are contained in the queue.
- **ExpFunction**: Manages the computation of $\exp(x)$ for large values of x , using `BigDecimal`; For large values of x , the standard `Math.exp(x)` cannot be used, since it only has double precision; Implements `pow()` for integer powers of a `BigDecimal` number and an approximation to the natural logarithm of a `BigDecimal` number.
- **ForceField**: Interface for different types of force fields (currently not used).
- **GenericResidueTemplates**: This class reads from a generic residue file that includes element type, AMBER atom type, limited connectivity, and partial charge. This file is analogous to `AminoAcidTemplates.java`; instead of amino acid parameters, parameters for general compounds and nucleic acids (referred to as ‘generic residues’) are read. By matching these generic residue templates to actual generic residues in a molecule, the corresponding template atom types and partial charges can be assigned to the matched residues. The format of the input parameter file is similar to the PARM AMBER datafiles, identical to the `all_amino94.in`.
- **KSParser**: The main class that sets up and handles the basic OSPREY computation and related functions. The OSPREY functions include: `doDEE` - perform DEE/ A^* redesign (this includes `MinDEE`, `BD`, and `BRDEE`); `genStructDEE` - generate structures for a selected set of the top `doDEE` conformations; `precomputeBackrubs` - precompute a list of allowed backrubs for each flexible residue position (used by `BRDEE`); `KSMaster` - perform K^* redesign; `doSinglePartFn` - generate (bound or unbound) structures for the K^* ensemble of a given

protein-ligand complex; doResEntropy - use SCMF to compute the residue entropy for each (non-Pro) residue in a protein.

- KStar: This is the main class for the KStar program; essentially just a wrapper for the KSParser class.
- MinDEEIntervals: Computes the single and pair interval terms in the MinDEE/BD/BRDEE criteria. This class is not used for traditional DEE.
- Molecule: Handles functions and data associated with molecules. Handles rotations/translations of (parts of) molecules. Manages the data associated with a molecule; handles changes to the molecule (e.g., coordinate changes, deletion or addition of residues, etc.). Determines the bond information for the molecule.
- MSAStar: Uses A* search for single or multiple mutation sequences simultaneously to return the minimum-energy conformation; each consecutive run returns the next lowest-energy conformation, in order.
- MSMinBounds: Performs two different operations, depending on the input parameters: 1) Applies the Bounds/MinBounds pruning criteria: computes a lower bound on the energy of all conformations that contain a given rotamer i_r , for each rotamer; 2) Computes: (a) a lower bound on the energy of all conformations that contain a pruned rotamer, and (b) all conformations that are pruned due to unallowed sterics.
- MutationManager: The MutationManager class maintains a list of mutations to be tested, maintains their scores, prints a log file, and generally manages the mutations to test.
- OneMutation: Handles the data for a single mutation sequence. Contains the amino acid identities for the given sequence and can contain the computed score. Implements a method for comparing two sequences that is used for sorting all sequences with respect to different criteria.
- ParamSet: Handles reading in and managing parameter/value pairs from the input configuration files.
- PEMHandler: Manages operations on the pairwise energy matrices.
- ProbeStericCheck: Implements a steric check for atom pairs based on the Richardsons' Probe approach.
- QueueNode: Handles the data for a single node in the A* queue.
- Residue: Handles functions and data associated with residues.
- RotamerLibrary: This class implements a rotamer library reader. It reads from an input file that contains rotamer information for amino acid types or other generic residues.
- RotamerSearch: This class provides a variety of tools and search algorithms for doing rotamer-based searching over molecular conformations. Contains functions for computing the pairwise energy matrices and for performing DEE/A* and K* (with different types of minimization) mutation searches. The functions in this class are typically called after the necessary setup in KSParser.java is performed.

- RotMatrix: This class implements rotation matrices.
- RyanComparable: Interface for comparing objects.
- RyanQuickSort: This class implements quick sort; sorted objects must implement RyanComparable.
- SamplingEEntries: Handles data storage for elements in the pairwise energy matrices.
- SimpleMinimizer: This class implements a simple energy minimization routine for the side-chains only. Handles the computation of the Amber dihedral energy penalties (for minimizing away from the initial rotamer dihedrals). Additionally there is a special residue, the ligand, that can translate and globally rotate.
- StericCheck: Implements functions for checking the steric overlap for specified parts of a given structure at different stages of the A* expansion of the conformation tree.
- Strand: Handles functions and data associated with strands.
- StrandRotamers: This class handles rotamer assignment and maintenance for a given strand. Performs rotamer swaps and amino acid mutations for this strand.
- VolModule: This class computes a crude molecular volume for a specified molecule.

Bibliography

- [1] C. Chen, I. Georgiev, A. C. Anderson, and B. R. Donald. Computational structure-based redesign of enzyme activity. *PNAS USA*, 106(10):3764–3769, 2009.
- [2] E. Conti, T. Stachelhaus, M. Marahiel, and P. Brick. Structural basis for the activation of phenylalanine in the non-ribosomal biosynthesis of Gramicidin S. *EMBO J.*, 16:4174–4183, 1997.
- [3] W. Cornell, P. Cieplak, C. Bayly, I. Gould, K. Merz, D. Ferguson, D. Spellmeyer, T. Fox, J. Caldwell, and P. Kollman. A second generation force field for the simulation of proteins, nucleic acids and organic molecules. *JACS*, 117:5179–97, 1995.
- [4] I. Davis, A. Leaver-Fay, V. Chen, J. Block, G. Kapral, X. Wang, L. Murray, W. Arendall, J. Snoeyink, J. Richardson, and D. Richardson. MolProbity: all-atom contacts and structure validation for proteins and nucleic acids. *Nucleic Acids Res.*, 35:W375–383, Jul 2007.
- [5] W. L. DeLano. The pymol molecular graphics system (2008). DeLano Scientific, Palo Alto, California, USA. <http://www.pymol.org>.
- [6] J. Desmet, M. De Maeyer, B. Hazes, and I. Lasters. The dead-end elimination theorem and its use in protein side-chain positioning. *Nature*, 356:539–542, 1992.
- [7] I. Georgiev and B. R. Donald. Dead-end elimination with backbone flexibility. *Bioinformatics*, 23(13):i185–94, 2007. Proc. International Conference on Intelligent Systems for Molecular Biology (ISMB), Vienna, Austria (2007).
- [8] I. Georgiev, D. Keedy, J. S. Richardson, D. C. Richardson, and B. R. Donald. Algorithm for backrub motions in protein design. *Bioinformatics*, 24(13):i196–204, 2008. Proc. International Conference on Intelligent Systems for Molecular Biology (ISMB), Toronto, Canada (2008).
- [9] I. Georgiev, R. Lilien, and B. R. Donald. Improved pruning algorithms and divide-and-conquer strategies for dead-end elimination, with application to protein design. *Bioinformatics*, 22(14):e174–183, 2006. Proc. International Conference on Intelligent Systems for Molecular Biology (ISMB), Fortaleza, Brazil (2006).
- [10] I. Georgiev, R. Lilien, and B. R. Donald. The minimized dead-end elimination criterion and its application to protein redesign in a hybrid scoring and search algorithm for computing partition functions over molecular ensembles. *J Comput Chem*, 29(10):1527–42, 2008.
- [11] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

- [12] R. Lab. The software page :: Kinemage website. <http://kinemage.biochem.duke.edu/software/index.php>. [Online; accessed 12 Mar 2009].
- [13] T. Lazaridis and M. Karplus. Effective energy function for proteins in solution. *PROTEINS: Structure, Function, and Genetics*, 35:133–152, 1999.
- [14] A. Leach and A. Lemon. Exploring the conformational space of protein side chains using dead-end elimination and the A* algorithm. *Proteins*, 33:227–239, 1998.
- [15] R. Lilien, B. Stevens, A. Anderson, and B. R. Donald. A novel ensemble-based scoring and search algorithm for protein redesign, and its application to modify the substrate specificity of the Gramicidin Synthetase A phenylalanine adenylation enzyme. *J Comp Biol*, 12(6–7):740–761, 2005.
- [16] S. M. Lippow, K. D. Wittrup, and B. Tidor. Computational design of antibody-affinity improvement beyond in vivo maturation. *Nat Biotechnol*, 25(10):1171–6, 2007.
- [17] S. C. Lovell, J. Word, J. Richardson, and D. Richardson. The penultimate rotamer library. *Proteins*, 40:389–408, 2000.
- [18] N. Pierce, J. Spriet, J. Desmet, and S. Mayo. Conformational splitting: a more powerful criterion for dead-end elimination. *J. Comput. Chem.*, 21:999–1009, 2000.
- [19] C. A. Voigt, S. L. Mayo, F. H. Arnold, and Z. G. Wang. Computational method to reduce the search space for directed protein evolution. *Proc Natl Acad Sci U S A*, 98(7):3778–83, 2001.
- [20] J. Wang, W. Wang, P. Kollman, and D. Case. Antechamber, an accessory software package for molecular mechanics calculations. *J. Mol. Graphics Model.*, 25:247–260, 2006.
- [21] S. Weiner, P. Kollman, D. Case, U. Singh, C. Ghio, G. Alagona, S. Profeta, and P. Weiner. A new force field for molecular mechanical simulation of nucleic acids and proteins. *J. Am. Chem. Soc.*, 106:765–784, 1984.