Chapter 15

Pairwise and Multimeric Protein–Protein Docking Using the LZerD Program Suite

Juan Esquivel-Rodriguez, Vianney Filos-Gonzalez, Bin Li, and Daisuke Kihara

Abstract

Physical interactions between proteins are involved in many important cell functions and are key for understanding the mechanisms of biological processes. Protein–protein docking programs provide a means to computationally construct three-dimensional (3D) models of a protein complex structure from its component protein units. A protein docking program takes two or more individual 3D protein structures, which are either experimentally solved or computationally modeled, and outputs a series of probable complex structures.

In this chapter we present the LZerD protein docking suite, which includes programs for pairwise docking, LZerD and PI-LZerD, and multiple protein docking, Multi-LZerD, developed by our group. PI-LZerD takes protein docking interface residues as additional input information. The methods use a combination of shape-based protein surface features as well as physics-based scoring terms to generate protein complex models. The programs are provided as stand-alone programs and can be downloaded from http://kiharalab.org/proteindocking.

Key words Protein–protein docking, Multiple-protein docking, Multimeric protein docking, Macromolecular docking, Protein–protein interactions, Protein–protein interface prediction

1 Introduction

Protein complexes are involved in many important processes in a living cell. In order to understand the mechanisms of these processes, it is necessary to solve the 3D structure of the protein complexes. Experimental techniques such as X-ray crystallography and nuclear magnetic resonance (NMR) have been used to solve the 3D structure of protein complexes, as shown in the large number of entries of complex structures in the Protein Data Bank (PDB) [1]. When protein complex structures have not been solved by experiments, it is possible to use computational tools to construct models of these complexes. A protein docking program takes two or more component protein structures as input and assembles

Daisuke Kihara (ed.), *Protein Structure Prediction*, Methods in Molecular Biology, vol. 1137,
DOI 10.1007/978-1-4939-0366-5_15,

them into 3D structure models of a protein complex. Input proteins can be either experimentally solved or computationally modeled structures using protein structure prediction programs such as the ones described in earlier chapters in this book.

Existing docking methods generate from a few hundred to thousands of candidate complexes, which are ranked by a score that indicates which models are more probable. Most of the existing docking methods deal with pairwise docking, where only two proteins are assembled [2–11]. A smaller number of methods perform docking of multiple protein structures, called multimeric or multiple-protein docking [12–17].

In this chapter we introduce three protein docking programs developed in our group. First, we show how to use LZerD (Local 3D Zernike Descriptor-based protein docking program) [11], our pairwise protein–protein docking program, to create protein complex structures from two individual proteins. It uses geometric hashing [18] for docking conformation search and a rotation invariant mathematical surface shape representation, the 3D Zernike Descriptors (3DZD) [19–22], as the main scoring term for evaluating docking poses. Next, we discuss PI-LZerD (Predicted Interface-guided LZerD) [23], which uses additional predicted protein interface information to guide conformation searches of pairwise protein–protein docking. PI-LZerD runs LZerD around the neighborhood of the provided predicted interface residues and further refines docking poses by running LZerD a second time. Finally, we present our multiple-protein docking program, Multi-LZerD [17, 24–26], which can assemble more than two proteins. In the first phase, pairwise docking predictions are generated for every possible pair of component proteins using LZerD. Then, multiple-protein complex structures will be generated by combining the pairwise docking predictions generated in the first phase. A genetic algorithm (GA) is used to explore the combinatorial space. After a configurable number of iterations, a final refinement step is applied to the structures.

The following sections provide instructions on how to use the LZerD protein–protein docking software. The readers are encouraged to refer to the original publications for more detailed descriptions of the algorithms and benchmark results of the programs.

2 Materials

The programs in the LZerD docking suite are available on the Kihara Lab website, http://www.kiharalab.org/proteindocking. The LZerD pairwise docking program is available as a compressed file (*lzerddistribution.tar.gz*) from the LZerD section of the webpage. Similarly, there is a section for PI-LZerD that has a link to

the necessary files for PI-LZerD named *PI-LZerD.tar.gz*. Multi-LZerD is available as *multilzerddistribution.tar.gz* from the *Multi-LZerD* section. All packages are intended to run on Linux machines.

Once the files are downloaded to your computer, they need to be decompressed. If a graphical file explorer is used, right clicking on the file and choosing a decompression option should usually extract them. If you are using a command line terminal, you can decompress each file by running *tar –zxf lzerddistribution.tar.gz* (for the LZerD package, or specify the corresponding file name for PI-LZerD and Multi-LZerD). Once files are decompressed, a new folder will be created with several programs in it. The details of each package's contents, their roles in the procedure, and input data are described next.

2.1 Pairwise Docking Package, LZerD

LZerD needs two files containing protein structures as inputs. The two input files should follow the PDB format (the format is described at http://www.wwpdb.org/docs.html). LZerD only requires the ATOM fields in the file. Below, whichever protein is provided first will be called *receptor*, while the second one is called *ligand*.

The final output consists of many PDB files that represent different poses of the ligand only, since LZerD scans for poses of the ligand around the receptor placed at the original position. This means that the best docking pose needs to be generated by combining the receptor PDB file and ligand1.pdb, the file that contains the best ranking pose. In the same way, the second best prediction can be generated with the receptor file and ligand2.pdb, and so on. Optionally, the user can re-rank a subset of the docking poses using a physics-based scoring function, described later in the chapter.

A prediction can be executed with a series of programs written in C/C++ and a Linux Shell script that triggers the main process:

- runlzerd.sh: The main script that receives two input PDB files and carries out the complete process by invoking a series of programs.
- mark_sur: An auxiliary program that marks residues on the protein surface. It uses the uniCHARMM file that is also included in the package.
- GETPOINTS and LZD32: These two programs create surface information used by LZerD to compare protein surface shapes.
- LZerD1.0: The main program that performs the pairwise docking.
- PDBGEN: A post-processing program for generating the best ligand poses obtained by LZerD1.0.
- lzerd_rerank.sh: Script used to re-rank the docking poses using a physics-based scoring function.

2.2 Predicted Interface-Guided Protein Docking Package, PI-LZerD

To run PI-LZerD, the user needs the atomic structures of a receptor and a ligand in the PDB format as well as the list of predicted protein interface residues. Protein interface residues can be predicted using a protein interface prediction method, such as BindML [27] or meta_PPISP server [28], or they can be provided based on biological knowledge of the proteins. The output files represent predicted complex conformations in the PDB format. The files will be found in the PI_LZerD/10.Result directory.

2.3 Multiple Docking Package, Multi-LZerD

Similar to the previous program, Multi-LZerD receives protein structure files that follow the PDB format. Since Multi-LZerD is for multiple-protein docking, the number of files will be three or more, not just two. The input protein structure files should have a common file name and a suffix, ".pdb". Also, a prefix should be associated with the files to indicate the chain ID of the units. For example, three input files for a trimeric protein complex can be named as follows: A-mytrimer.pdb, B-mytrimer.pdb, and C-mytrimer.pdb. In this case the common file name is *mytrimer* and the prefixes are A, B, and C. In addition, it is required that the chain ID is provided in each ATOM field. We suggest that the same chain ID be used as the prefix.

The program outputs a series of PDB files. refined-00001.pdb is the best scoring model, refined-0002.pdb the second best, and so on. A set of decoy-<number>.pdb files are also created that represent the predicted structures before the refinement step (details described later). Multi-LZerD output files are complete models; i.e., each file contains all protein units in their predicted poses. This differs from LZerD where only the *ligand* part of the prediction is generated as output.

Since Multi-LZerD is a superset of LZerD, the programs mentioned in the previous LZerD section are also used here. The following list describes the additional scripts and programs that are incorporated for multiple docking:

- run.sh: The main script that executes the complete protocol. It is necessary to use proper suffix and prefixes for input file names before executing it.
- addhydrogens.pl: Given that we do not assume that the PDB files contain hydrogen atoms, this script will computationally add their atomic coordinates. This is required because of the scoring function used in Multi-LZerD.
- create_lzerd_decoys.pl and multilzerd_sort_decoys.pl: These two scripts trigger the pairwise docking performed by LZerD.
- multilzerd_pairwise_cluster.pl: The program performs clustering of pairwise docking predictions from LZerD to eliminate poses that are too similar.
- multilzerd: The main multiple protein docking program.

- multilzerd_create_pdb: Creates the decoy-<number>.pdb files that represent the final multiple docking poses.
- multilzerd_refine: An additional post-processing step that tries to improve the final multiple-protein docking poses.

3 Methods

This section describes steps to run the three docking programs: LZerD, PI-LZerD, and Multi-LZerD.

3.1 LZerD

The LZerD protocol flow chart is shown in Fig. 1. LZerD takes two PDB files, which will be docked with each other, as input.

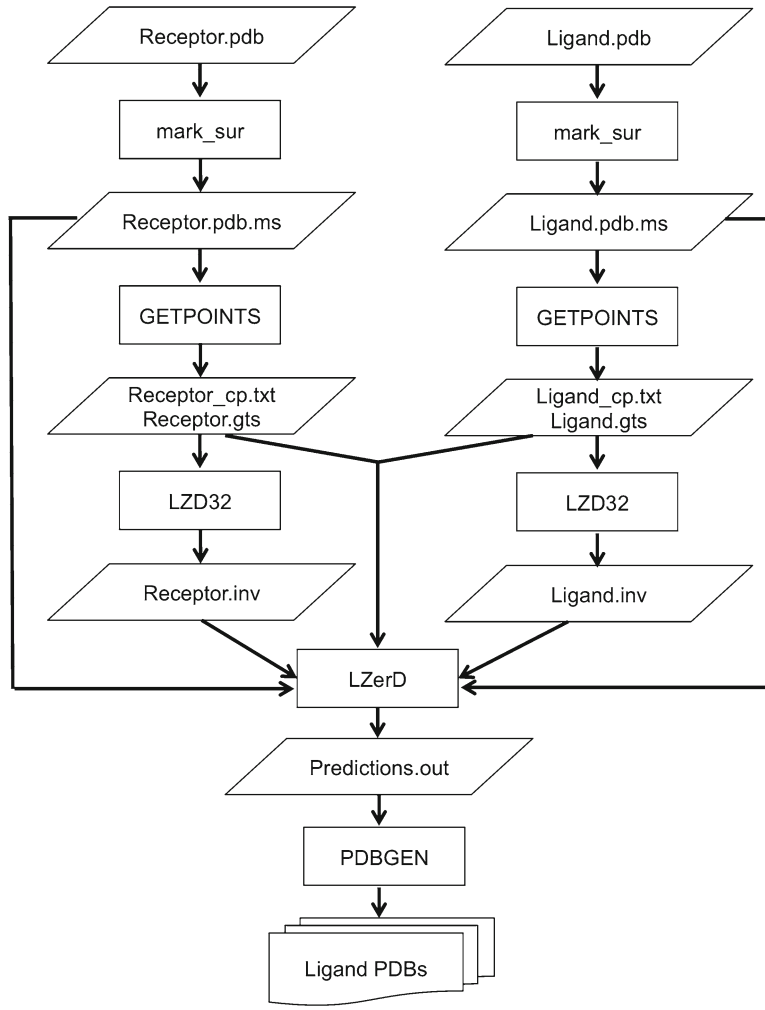

Fig. 1 LZerD flow diagram. Two input PDB files are first pre-processed using mark_sur, GETPOINTS and LZD32. Then, the intermediate files are fed into LZerD to create the pairwise docking predictions

They are called the receptor and ligand protein. The package contains a folder called *example*, which contains a sample receptor protein (1PPE_r_b.pdb) and a sample ligand (1PPE_l_b.pdb) protein file as well as all output files. We will use these two files to illustrate the procedure to perform protein docking with LZerD. We first show how to run the main script from a command line using default program settings. In the following sections, we explain each step in the process, intermediate files output by the programs, and possible parameter modifications users can control.

3.1.1 Main LZerD Process

In order to execute LZerD, users first need to locate the folder where the files were downloaded. For example, if the package was saved to a "Downloads" folder in the user's home directory, open a command terminal window and input the command

```
cd Downloads/lzerddistribution
```

Start the process by running the main script, *runlzerd.sh*, with two input PDB files as follows:

```
./runlzerd.sh example/1PPE_r_b.pdb example/1PPE_l_b.pdb
```

Here the sample receptor and ligand files provided in the example folder are used, but they can be any two PDB files. The process can run for several hours before reaching completion depending mainly on the size of the proteins. Some of the parameters used internally in the script will also influence how much computation time is required.

While the reader can change parameters in LZerD, simply running the shell script as shown above will complete the computation and produce candidate docking models. In the case of the example, the best model generated by LZerD will be composed of *1PPE_r_b.pdb* and *ligand1.pdb*. If the user requires a single PDB file, the two files can be concatenated by the *cat* command:

```
cat example/1PPE_r_b.pdb ligand1.pdb>model1. pdb
```

The new file *model1.pdb* can be visualized with a standard protein structure viewer such as PyMol or RasMol.

3.1.2 Detailed Steps

In the following sections, we provide more detailed information about each step in *runlzerd.sh* including parameters users can change.

Surface calculation. LZerD constructs the surface for two input protein structures in order to identify surface shape complementarity at the interface regions. This is the main term in the docking scoring function. First, it uses *mark_sur* to identify amino acid residues on the protein surface and adds annotations to the protein atoms in the ATOM fields of the PDB file. This will generate *.pdb.ms* files from the input *.pdb* files. In our example, *1PPE_r_b.pdb.ms* and *1PPE_l_b.pdb.ms* are the intermediate files created by this step:

```
./mark_sur 1PPE_r_b.pdb 1PPE_r_b.pdb.ms
./mark_sur 1PPE_l_b.pdb 1PPE_l_b.pdb.ms
```

Using marked surface residues, GETPOINTS creates two intermediate surface representations: an isosurface stored in *.gts* files and a point representation stored in *_cp.txt* files. The "-cut" parameter given to GETPOINTS is the key to control the execution time of the pairwise docking program as shown here:

```
./GETPOINTS -pdb 1PPE_r_b.pdb.ms -smooth 0.35 -cut 1e-04
./GETPOINTS -pdb 1PPE_l_b.pdb.ms -smooth 0.35 -cut 1e-04
```

This value is related to the precision of the surface representation used: lower values represent more fine-grained details and precision. The package downloaded contains a value of 1e-04, which we recommend for higher quality results with a long computation time. In contrast, using 1e-02 would yield a coarse-grained conformation sampling, which may be useful for obtaining preliminary results faster. A compromise between the time and the accuracy would be obtained by using a value of 1e-03.

Next, LZerD creates fingerprint representations of the shape around the surface points determined by *GETPOINTS*. These are stored in files with the *.inv* suffix. A fingerprint of a surface point is a vector of 36 numbers by default, which are coefficient values of a rotation invariant descriptor called the 3D Zernike descriptor [11, 22]. The *LZD32* program computes the fingerprints:

```
./LZD32 -g 1PPE_r_b.gts -c 1PPE_r_b_cp.txt -o 1PPE_r_b -dim 161 -rad 6.0 -ord 10
./LZD32 -g 1PPE_l_b.gts -c 1PPE_l_b_cp.txt -o 1PPE_l_b -dim 161 -rad 6.0 -ord 10
```

Please refer to our previous publications [21, 22] for more details about the 3D Zernike descriptors. The "-ord" parameter specifies the order of the 3D Zernike descriptors, which determines the number of elements in the vector. We found in our work [11] that 10 is an appropriate value, but the user could modify the order. For example, changing the order from 10 to 20 will increase the vector sizes from 36 to 121, which makes the fingerprint represent more details of the surface shape.

Pairwise docking. Using the files prepared in the previous step, LZerD1.0 will create protein–protein docking predictions:

```
./LZerD1.0 -rec 1PPE_r_b_cp.txt -lig 1PPE_l_b_cp.txt -prec
1PPE_r_b.pdb.ms -plig 1PPE_l_b.pdb.ms -zrec 1PPE_r_b_01.inv -zlig
1PPE_l_b_01.inv -rfmin 4.0 -rfmax 9.0 -rfpmax 15.0 -nvotes 8 -cor 0.7
-dist 2.0 -nrad 2.5 > 1PPE_r_b_1PPE_l_b. out
```

The output will be added to the file *1PPE_r_b_1PPE_l_b.out* progressively as the protein docking program analyzes different combinations of matching regions. The number of models generated can be obtained by counting the lines in the file, by running *wc* (word count):

```
wc 1PPE_r_b_1PPE_l_b.out
```

In a LZerD output file, each prediction is represented as a rotation and translation of the ligand protein from its original position. One line in the file is composed of 12 numbers, representing a transformation matrix, and an additional value that holds the score, for instance:

```
0.174 -0.868 0.465 -0.948 0.275 0.160 -0.267 0.413
-0.871 37.305 5.810 -2.871 397.116
```

In this example, the score is 397.116. The higher the score, the better the shape complementarity at the docking interface is, which indicates a better model.

In addition to this type of row it is possible that the file has a single header row like the following:

```
LIG: 0.926583 0.541623 2.32135 -1.96904 -8.93765 -2.37821
```

This is an optional random transformation that is applied to the ligand before beginning the docking process. This can be triggered by providing the –randomize flag when executing LZerD.

Generation of poses in PDB format. The number of models to output is one of the key parameters users can modify in *runlzerd.sh*. A program called PDBGEN is in charge of creating the different ligand docking poses in PDB format by receiving the receptor PDB, the ligand PDB, the output file, and the number of models to be created.

In the example case, PDBGEN is run as follows:

```
./PDBGEN 1PPE_r_b.pdb 1PPE_l_b.pdb 1PPE_r_b_1PPE_l_b.out 3
```

This generates three ligand files (ligand1.pdb, ligand2.pdb, and ligand3.pdb). The last number, 3, is the number of ligand models to generate, which can be increased to provide more models. PDBGEN can generate any number of models from one up to the number of lines in the ".out" file. As a reminder, only the ligand part of the complex is output in this step. To have a single PDB file with both receptor and ligand it is necessary to join both files.

Re-ranking LZerD predictions. The order in which rows appear in the *.out* file is based on the shape agreement between different docking poses. Optionally, users can re-rank the predictions based on a different type of score, a physics-based scoring function, described in one of our previous publications [17]. It considers van der Waals and electrostatic potentials, hydrogen and disulfide bonding, solvation, and additional knowledge-based contact terms. This is a more computationally intensive scoring method compared to the shape-based function used by LZerD. Thus, we recommend that this only be applied to a subset of the predictions in the *.out* file. To do so, the user needs to run the following script:

```
./lzerd_rerank.sh 1PPE_r_b.pdb 1PPE_l_b.pdb 1PPE_r_b_1PPE_l_b.out 100
```

This will generate a comma-separated file, called 1PPE_r_b_1PPE_l_b.out.reranked, with 100 text lines each representing a new score assigned to each of the first 100 predictions contained in the original *.out* file. This is a sample result that contains the first three lines of a re-ranked file:

```
model1.pdb,-9825.92,52,-69.0978,2112.1,
-19.1428,-1041.4,-207.964,1030.63,748.893,
-5.22232,-1403.18,-123.551,-14.0836
model21.pdb,-8636.75,41,-55.1373,1215.62,
-11.8204,-819.328,-189.976,805.575,665.787,
-2.24271,-1322.89,-123.589,-9.36367
model7.pdb,-6880.77,35,-43.6754,2579.88,
1.1024,-577.526,-111.942,554.141,252.346,
-4.48198,-1070.18,-123.285,-1.538
```

For example, the first line indicates that model1 (column 1) from the original *.out* file also has the best score using the physics-based function (column 2). However, we can see that the second best model, according to this new scoring function, comes from line 21 (model21) in the original *.out* file. The rest of the columns in the file represent the individual terms in the scoring function. Among them only the second column is used for re-ranking purposes. In this scoring function, a lower value represents a better score.

3.2 PI-LZerD

The execution of PI-LZerD takes a considerably longer time to finish compared to LZerD, because it runs pairwise docking multiple times. The PI-LZerD package consists of several programs, which are all combined in a shell script. First we present the shell script along with a reference to example files available online. Then, we discuss each step in the process and the estimated computational time required for the steps.

3.2.1 Main PI-LZerD Process

A single script, *PI_LZerD.sh*, runs the entire procedure from the beginning to the end. *PI_LZerD.sh* sets global variables based on user input and goes into each subdirectory to run individual programs sequentially. Each individual job receives parameters through global variables. For each job, *PI_LZerD.sh* checks the results and reports errors when necessary.

A detailed example is provided on the website using the protein complex (PDB ID: 1A2K) as an example (http://www.kiharalab.org/proteindocking/PI-LZerD/example). In this example, the input files are named 1A2K_R.pdb and 1A2K_L.pdb after their chain IDs, R and L, which are denoted in the PDB file. In addition, (predicted) docking interface residues should be provided. The files that contain the interface predictions in the example are called 1A2K_R.meta and 1A2K_L.meta. The user only needs to execute the following command in a terminal to run the complete PI-LZerD protocol:

```
./PI_LZerD.sh 1A2K R L
```

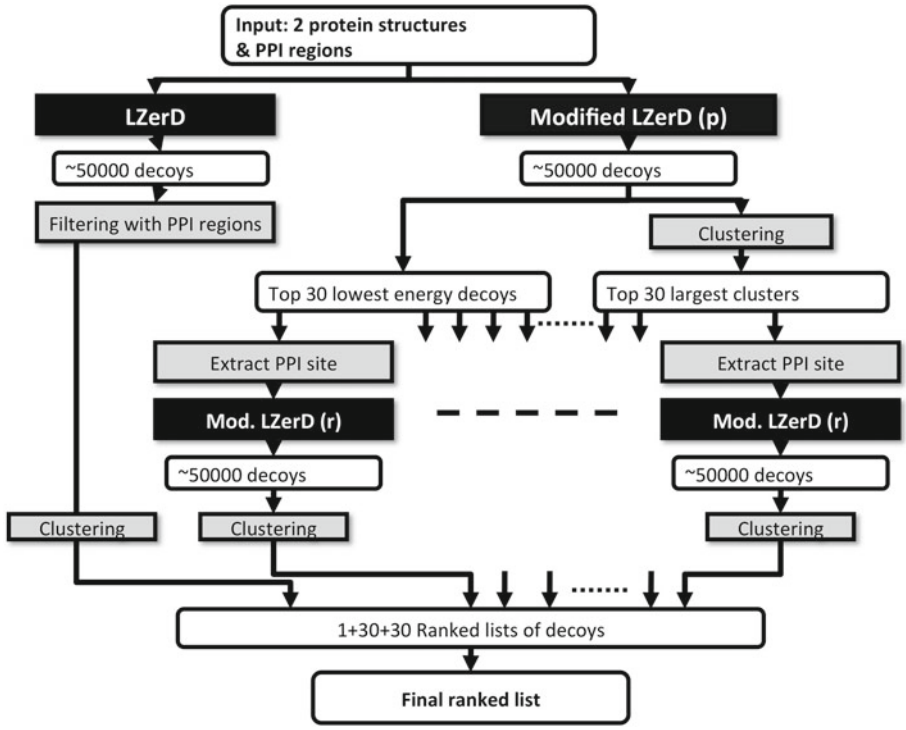

Fig. 2 Overview of the PI-LZerD algorithm. On the left branch the original LZerD program is run. LZerD (p) uses the permissive search space, and LZerD (r) uses restrictive search space employed in the geometric hashing stage. Refer also to the original PI-LZerD paper [23]

3.2.2 Detailed Steps

The flow chart of the PI-LZerD algorithm (Fig. 2) is provided in the original PI-LZerD paper [23]. PI-LZerD is computationally expensive, because it runs LZerD pairwise docking 61 times in an iterative fashion. Thus, the total running time can be over a week, if executed sequentially. The 60 independent runs of LZerD in the second iteration can be distributed across multiple CPUs in a single machine or in a computing cluster and run in parallel.

The PI-LZerD package is organized in a folder structure that reflects computational steps (Fig. 3). There are 16 major steps in total as shown in Fig. 4, which include 23 sub-steps listed in Table 1. For each folder, a script named *job.sh* is used to run the corresponding programs. The *job.sh* scripts communicate with the main script, *PI_LZerD.sh*, using the global environment with the main script running the scripts in each directory following the numeric order. It also checks the return values from each script and detects if any errors occurred. As shown in Fig. 4, the whole procedure can be divided into five stages.

Stage 1. LZerD run, without interface residue information: LZerD is run without using protein interface information. This stage corresponds to the leftmost branch in Fig. 2. The LZerD program used in PI-LZerD has been modified from the original LZerD

program. The modified LZerD outputs matched critical points from two proteins in each predicted conformation. Files are compressed and saved to a file called <PDB_ID> .out.gz. This information is later used for both the permissive LZerD and restrictive LZerD

```
LZerD/                    PI_LZerD
|-- 01.BASE               |-- 01.Pred_PPI
|   |-- CP                |   |-- 01.Pred_PPI_pdb
|   |-- INV               |   |-- 02.PPI_CP
|-- 02.MPI                |   |-- 03.PPI_RES
|   |-- 01.LZerD_MPI      |   `-- 04.Res_Dist
|   |-- 02.LZerD_MAT      |-- 02.Sim_LZerD
|   |-- 03.SRB_MPI        |-- 03.LZerD_CIRMSD
|   `-- 04.IRMSD_MPI      |-- 04.CIRMSD_CLUST
|-- 03.Votes              |-- 05.CLUST_T60
|-- 04.ORD_SRB            |-- 06.SRF
|-- 05.CI_RMSD            |-- 07.T60_Iter
`-- 06.LZerD_CIRMSD       |-- 08.K60_CIRMSD
                          |-- 09.T61
                          `-- 10.Result
```

Fig. 3 PI-LZerD directory structure. There are two directories in PI-LZerD: LZerD and PI_LZerD. Under each directory, there is a script named job.sh, which is sequentially called by the main script PI_LZerD.sh

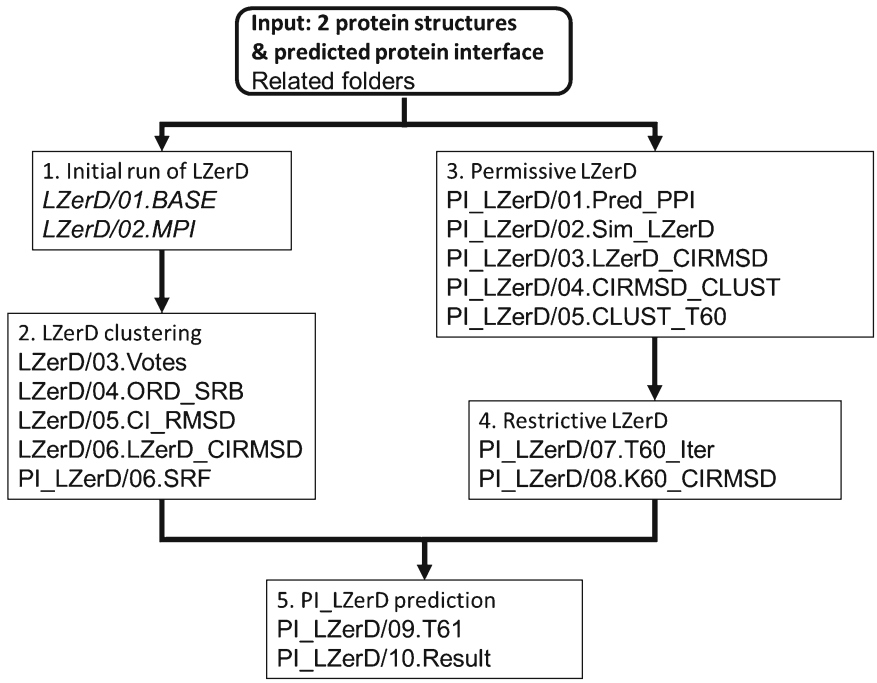

Fig. 4 Computing steps in PI-LZerD. Five steps of PI-LZerD and programs included in the steps

Table 1
Detailed steps of PI-LZerD. All 23 steps in PI-LZerD are summarized using 1A2K as an example of input. A brief description and input and output files are provided

Stage	Step	Description	Input	Input example	Output	Output example
1	LZerD/ 01.BASE	Add hydrogen atoms, find and mark the surface residues using mark_sur	Receptor and ligand protein structure files	1A2K_R.pdb, 1A2K_L.pdb	Receptor and ligand protein structures with flagged surface residue information	1A2K_R.pdb, 1A2K_L.pdb
1	LZerD/ 01.BASE/CP	Generate critical point information using GETPOINTS	Receptor and ligand proteins with surface residue information	1A2K_R.pdb, 1A2K_L.pdb	Critical pint (.cp) files	1A2K_R.gts, 1A2K_R.cp, 1A2K_L.gts, 1A2K_L.cp
1	LZerD/ 01.BASE/INV	Compute 3D Zernike descriptors for each critical point	gts and critical point (.cp) files	1A2K_R.gts, 1A2K_R.cp, 1A2K_L.gts, 1A2K_L.cp	Zernike descriptors in inv format	1A2K_R.inv, 1A2K_L.inv
1	LZerD/ 02.MPI/ 01.LZerD_MPI	Initial run of protein docking prediction using LZerD	Receptor and ligand structure, critical point information, Zernike descriptors	1A2K_R.pdb, 1A2K_L.pdb, 1A2K_R.gts, 1A2K_R.cp, 1A2K_L.gts, 1A2K_L.cp	Initial run of protein docking predictions	1A2K.out.gz
1	LZerD/ 02.MPI/ 02.LZerD_MAT	Extract rotation matrices information for physics-based scoring and IRMSD computation	Extract rotation metrics information for physics-based scoring and IRMSD computation	1A2K.out.gz	Extracted rotation/ translation matrices information	1A2K.mat
1	LZerD/ 02.MPI/ 03.SRB_MPI	Compute physics-based score for each prediction	Rotation/translation matrices, and the receptor–ligand protein structure	1A2K.mat	Physics-based scores	1A2K.srb

1	LZerD/02.MPI/04.IRMSD_MPI	Compute physics-based interface RMSD (IRMSD) for each prediction	Rotation/translation matrices, and the receptor–ligand protein structure	1A2K.mat	Interface RMSDs	1A2K.irmsd
1	LZerD/03.Votes	Extract the critical point dependencies	Initial protein docking prediction result	1A2K.out.gz	The critical point dependency information	1A2K.Votes
2	LZerD/04.ORD_SRB	Sort predictions by physics-based scores	Initial protein docking prediction result	1A2K.srb, 1A2K.irmsd	Sorted predictions	1A2K.LZerD
2	LZerD/05.CI_RMSD	Compute pairwise Common Interface RMSD (CI_RMSD) for top 1,000 predictions	Initial protein–protein docking predictions	1A2K.LZerD	Common Interface RMSD (CI_RMSD) for top 1,000 predictions	1A2K.cirmsd
2	LZerD/06.LZerD_CIRMSD	Cluster predictions based on CI_RMSD	Top 1,000 predictions	1A2K.LZerD, 1A2K.cirmsd	Clustered top 1,000 predictions	1A2K.t1k
2	PI_LZerD/06.SRF	Use naïve-filtering method on predicted interface residues	Top 1,000 predictions from first iteration LZerD program	1A2K.LZerD, res.txt	Top 1,000 predictions sorted by the percentage of agreement with the predicted interface residues	1A2K.rcf
3	PI_LZerD/01.Pred_PPI/01.Pred_PPI_pdb	Protein interface prediction results from meta-PPISP server	Receptors and ligand structure files	1A2K_R.meta, 1A2K_L.meta	Predicted protein interface	1A2K_R.rec.pdb, 1A2K_L.lig.pdb
3	PI_LZerD/01.Pred_PPI/02.PPI_CP	Compute the critical points belonging to predicted interface residues	Receptor/ligand structure and predicted protein interfaces	1A2K_R.cp, 1A2K_L.cp, 1A2K_R.rec.pdb, 1A2K_L.lig.pdb	Critical points on predicted interface residues	1A2K_R.rec.pdb, 1A2K_L.lig.pdb

(continued)

Table 1
(continued)

Stage	Step	Description	Input	Input example	Output	Output example
3	PI_LZerD/01.Pred_PPI/03.PPI_RES	Compute the predicted interface residues	Predicted protein interface	1A2K_R.cp, 1A2K_L.cp, 1A2K_R.rec.pdb, 1A2K_L.lig.pdb	List of predicted interface residues	res.txt
3	PI_LZerD/02.Sim_LZerD	Permissive LZerD using predicted interface residues	Receptor and ligand structure and list of predicted interface residues	1A2K.Votes, res.txt	First iteration LZerD prediction	1A2K.sim
3	PI_LZerD/03.LZerD_CIRMSD	Compute pairwise CI_RMSD distances on top 1,000 predictions	Top 1,000 predictions from initial run of LZerD prediction	1A2K.sim	CI_RMSD distances of top 1,000 predictions	1A2K.cirmsd
3	PI_LZerD/04.CIRMSD_CLUST	Cluster top 1,000 predictions based on CI_RMSD distances	Top 1,000 predictions from initial run of LZerD program	1A2K.cirmsd	Clustering from top 1,000 predictions	1A2K.t1k
3	PI_LZerD/05.CLUST_T60	Select top 60 clustered predictions	Clustered top 1,000 predictions	1A2K.t1k	Selected top 60 clustered predictions	1A2K.40A.t60
4	PI_LZerD/07.T60_Iter	Restrictive LZerD using 60 clustered prediction	Selected top 60 clustered predictions	1A2K.40A.t60	Restrictive LZerD prediction based on the 60 clustered predictions	1A2K.iter.gz, 1A2K.k60
4	PI_LZerD/08.K60_CIRMSD	Pairwise CI_RMSD distances on top 1,000 predictions for 60 prediction lists	Top 60 clustered predictions	1A2K.k60	60× Pairwise CI_RMSD distances	1A2K.cirmsd
5	PI_LZerD/09.T61	Merge lists of 61 predictions	Selected top 60 clustered predictions	1A2K.cirmsd, 1A2K.LZerD	Re-ranked predictions	1A2K.i61
5	PI_LZerD/10.Result	Generate complex conformation for each prediction	Merged prediction list	1A2K.i61	Complex conformation for each prediction	1A2K.1.pdb

process (the right branch in Fig. 2). A score is computed each predicted conformation with a physics-based scoring function, the same function used in the LZerD pairwise docking. Folders for Step 1 and their roles are as follows:

Related folders:

```
LZerD/01.BASE Add hydrogen atoms, compute critical points and 3DZD for input PDB files
LZerD/02.MPI  Initial run of LZerD
LZerD/03.Votes Compute matching critical points for each prediction
LZerD/04.ORD_SRB Computing physics-based score
```

Stage 2. Clustering docking models: The top 1,000 scoring docking models computed in Stage 1 are clustered. The similarity of docking models is measured with the Common Interface RMSD (CI_RMSD), which is the RMSD computed only for residues that are common in the models to be compared. Then, the representatives of the clusters are sorted by considering the agreement of the docking interface residues to the provided predicted protein interface information (naïve-filtering method). This corresponds to the clustering of the leftmost branch in Fig. 2.

Related folders:

```
LZerD/05.CI_RMSD          Compute CI_RMSD
LZerD/06.LZerD_CIRMSD     Clustering by CI_RMSD
PI_LZerD/06.SRF     Naïve-filtering method
```

Stage 3. LZerD run with provided docking interface residue information (permissive search). Next, the process moves to the right branch in Fig. 2. LZerD is run to explore docking conformations in the vicinity of the provided docking interface residue information. More concretely, the interaction interface of docking poses needs to have some overlap with the provided interface residues. The permissive search means that the conformation search space is set to be larger compared to the next iteration of LZerD runs (Stage 4). 50,000 docking models are generated and clustered in terms of CI_RMSD.

Related folders:

```
PI_LZerD/01.Pred_PPI      Predicted protein interface information
PI_LZerD/02.Sim_LZerD     Running Permissive LZerD
PI_LZerD/03.LZerD_CIRMSD  CI_RMSD of permissive LZerD results
PI_LZerD/04.CIRMSD_CLUST  Clustering of permissive LZerD results
```

```
   PI_LZerD/05.CLUST_T60     Selecting  Top  60
cluster selection
```

Stage 4. Second LZerD run with interface residue information (restrictive search): Among the clustering results of the previous LZerD (Stage 3), the 30 largest clusters are selected and the docking model that is closest to the cluster centroid in each cluster is kept. In addition, the 30 lowest energy docking models are also selected (the energy values of all docking models have already been computed and stored in Stage 1). Thus, a total of 60 docking models are kept from Stage 3. For each of the 60 docking models, docking interface residues are extracted and considered as updated predictions of interface residues. Then, LZerD is run using the updated interface residue information similar to Stage 3, but this time the conformational search is restricted to a smaller area close to the provided interface residues. Docking models are clustered based on CI_RMSD, and the one model that is closest to the cluster centroid is kept.

Related folders:

```
   PI_LZerD/07.T60_Iter      Restrictive LZerD
   PI_LZerD/08.K60_CIRMSD    Clustering    with
CI_RMSD for restrictive LZerD results
```

Stage 5. Generating PI-LZerD prediction: At this stage there are 61 ranked lists of docking models, each from an independent LZerD run. The last step of PI-LZerD is to compute the final ranked list of predictions out of the 61 lists. From each of the 61 lists, the top-ranked predictions are first ranked among themselves by their scores. Then, the second-ranked predictions from each file are ranked in the same way. This is repeated for predictions at the same subsequent ranks in the files. Thus, the final output is a list of predictions in the 61 lists, which are first sorted by their ranks and then sorted by the scores. Finally, the conformations of the top predictions are then generated.

Related folders:

```
   PI_LZerD/09.T61      61 prediction lists
   PI_LZerD/10.Result  Merged   61   prediction
lists
```

Table 2 lists the expected running time for each step. Generally a "fast" process can be finished in minutes, a "medium" process should be finished in hours, and a "slow" process can take days to finish.

3.3 Multi-LZerD

The Multi-LZerD package contains a script, *run.sh*, that runs the complete process of multiple-protein docking. The overall steps are illustrated in Fig. 5. To execute *run.sh*, the names of the input files need to be specified as explained in the next section. The later sections will go into more details about the intermediate steps performed by *run.sh*.

Table 2
Running time for steps in PI-LZerD. Fast indicates minutes, medium indicates hours, and slow processes can take days to finish

Stage	Folder	Time	References
1	LZerD/01.BASE	Medium	Add hydrogen atoms, create critical points, compute Zernike descriptors
1	LZerD/02.MPI/01.LZerD_MPI	Slow	Original LZerD program with modified output
1	LZerD/02.MPI/02.LZerD_MAT	Medium	Extract prediction matrix
1	LZerD/02.MPI/03.SRB_MPI	Slow	Compute the physics-based scores
1	LZerD/02.MPI/04.IRMSD_MPI	Slow	Compute the interface RMSD
1	LZerD/03.Votes	Medium	Extract the critical point dependence information
2	LZerD/04.ORD_SRB	Medium	Sort predictions in physics-based scores
2	LZerD/05.CI_RMSD	Medium	Compute common interface RMSDs (CI-RMSD)
2	LZerD/06.LZerD_CIRMSD	Medium	Generate list of clustered original LZerD predictions
2	PI_LZerD/06.SRF	Medium	Apply simple residue filtering method
3	PI_LZerD/01.Pred_PPI/	Fast	Extract protein interface information
3	PI_LZerD/02.Sim_LZerD	Slow	LZerD(p)
3	PI_LZerD/03.LZerD_CIRMSD	Medium	Compute CI-RMSDs for LZerD(p)
3	PI_LZerD/04.CIRMSD_CLUST	Medium	Cluster based on CI-RMSD
3	PI_LZerD/05.CLUST_T60	Medium	Generate top 30 + 30 predictions
4	PI_LZerD/07.T60_Iter	Slow	Run LZerD(r) for 60 predictions
4	PI_LZerD/08.K60_CIRMSD	Medium	Compute CI-RMSD for 60 predictions
5	PI_LZerD/09.T61	Fast	Generate list from 60 + 1 lists of predictions
5	PI_LZerD/10.Result	Fast	Generate complex conformations

3.3.1 Main Shell Script for Multi-LZerD

Multi-LZerD requires that input file names follow a specific naming convention. As an example, we provide three protein files in the package: A-sample.pdb, B-sample.pdb, and C-sample.pdb. Every file name must start with a prefix which indicates the chain ID of the protein. We recommend using the same prefix as the chain

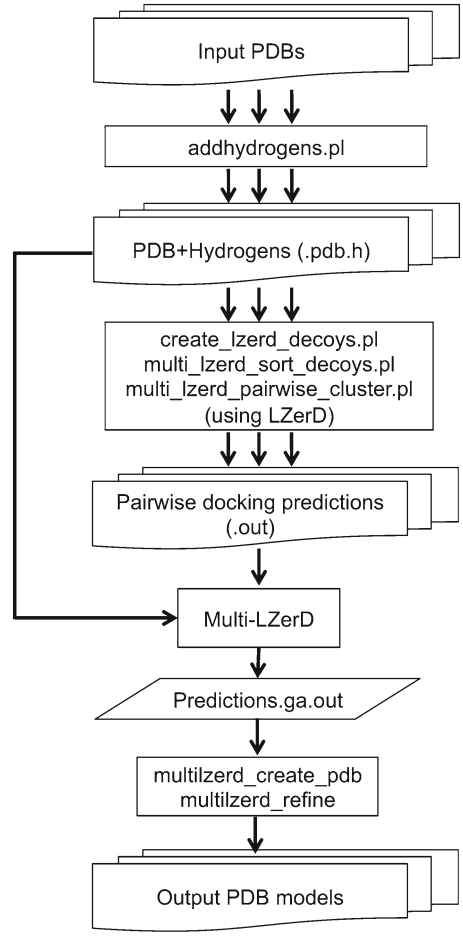

Fig. 5 Multi-LZerD flow diagram. Several PDB files, each representing a protein unit, are pre-processed to add hydrogen information and to generate pairwise docking predictions using LZerD. Multi-LZerD then combines the pairwise predictions and generates PDB files that contain the multimeric protein models

identifier in the PDB file of the protein. The prefix should be followed by a "-" and a base name (e.g., PDB file name, e.g., 1abc or *sample* in the case of the sample files) and a suffix, *.pdb*.

Once the input PDB file names are modified following the rule above, the file names should be specified in *run.sh* as follows:

```
basename = 'sample'
units = 'A,B,C'
```

Notice that the "-" and the suffix *.pdb* are not included in either the unit prefixes or as base name. Also, the unit prefixes must be comma separated. If the names of the files were *D-complex.pdb*, *E-complex.pdb*, and *F-complex.pdb*, they would be specified as follows:

```
basename = 'complex'
units = 'D,E,F'
```

Once the input files are specified in *run.sh*, users can run it by opening a terminal window and navigating to the folder where *run.sh* is located by using the *cd* command, and then, simply run it by typing:

```
./run.sh
```

When the computation is finished, a series of files with names starting with "refined" are created, which contain predicted protein complex models. The associated number indicates the rank of the models, with lower numbers representing better predictions.

3.3.2 Detailed Steps in Multi-LZerD

Here we explain the steps in *run.sh*. Parameters in some steps can be modified.

1. *Addition of hydrogen atoms.* The scoring function used in Multi-LZerD requires the presence of hydrogen atoms in protein structures. A script named *addhydrogen.pl* takes PDB files and generates a new set of files with a ".pdb.h" suffix:

```
./addhydrogens.pl sample A,B,C
```

   The newly created files contain all the original atoms and, additionally, hydrogen atom locations calculated using the HBPLUS program [29]. If users prefer to add hydrogen atoms in an alternative way, this step should be removed from *run.sh*. Please note that the files with hydrogen atoms should have a file name with *.pdb.h* at the end.

2. *Pairwise docking predictions.* Multi-LZerD uses LZerD to create pairwise poses between all possible combinations of pairs of component proteins. This means that LZerD will be executed several times. The following section in *run.sh* manages the necessary calls to LZerD to create pairwise docking predictions:

```
./create_lzerd_decoys.pl sample A,B,C execute
```

   create_lzerd_decoys.pl will terminate once all pairwise predictions are finished. Then, the top predictions for each case are kept using the following auxiliary script:

```
./multilzerd_sort_decoys.pl ./A,B,C 54000
```

   The number of pairwise poses kept in this example is 54,000, although users can change this value. A lower number will reduce the search space of complex structures and the execution time but may eliminate correct predictions. A higher number will increase the chance of finding correct and near-correct poses but at the price of increasing the execution time. These steps will produce result files that contain pairwise docking predictions for each pair. The file names will be, for

the example case, *A-B.out*, *A-C.out*, and *B-C.out*. In general, there will be a "*.out*" file for every pair of proteins identified by their prefixes.

3. *Removal of similar pairwise predictions.* By default, the protocol generates 54,000 pairwise poses per protein pair, but a number of them may be similar to each other. To minimize redundancy we apply a clustering program to group similar poses and keep one representative pose from every group. Predictions are grouped together if their root mean square deviation (RMSD) between C-α atoms is less than 10 Å, in the default case. Changing the following line in *run.sh* modifies this threshold:

```
cluster_threshold = 10
```

More pairwise poses will be kept with a smaller value. Conversely, a higher value will reduce pairwise docking conformations and the execution time more aggressively but with a risk of pruning out correct or near-correct models.

4. *Multiple-protein docking.* The main multiple-protein docking is performed by the following line in *run.sh*:

```
./multilzerd --pdbid sample --chains A,B,C
-o sample --generations 200
--population 200 --clashes 300 --cluster 10
--weights all
```

Users may want to change four parameters provided to the program, depending on the protein complexes they want to assemble.

5. *Number of generations.* The generation parameter, which is set to `--generations 200` as default, represents the number of iterations used to explore different pairwise prediction combinations during the GA performed by Multi-LZerD. More conformations will be explored with a higher number of iterations. The amount of iterations required varies for different cases. In our papers, we have explored the effect that the number of generations has in the final prediction accuracy [17, 24]. For cases up to six chains we observed that high-quality models were obtained within 2,000–3,000 iterations. Notice that the script sets the number of iterations to 200. We recommend this number to be increased if users can bear a longer computation time.
6. *Population size.* The population size, set as `--population 200`, is the number of conformations generated and compared in each generation of GA. A larger population size can explore more conformations. Obviously, the execution time will increase as the population increases. We have analyzed how much the population size impacts the quality of the models in our previous publication [24]. For most cases we recommend a population size between 200 and 400.

7. *Number of atom clashes allowed*. The atom clash parameter (--clashes), which is set to 300 as default, determines the number of atoms that we allow to be closer than 3 Å to another atom in a protein complex model. In the default setting, if 300 or more atoms in a model have clashes, the model is removed. When a target protein complex has a larger number of subunits, more clashes may be tolerated. The default setting of 300 clashes is appropriate for a smaller number of chains (e.g., 3 or 4). We have found that it needs to be increased to around 800–1,000 for a six-chain complex.
8. *Multiple-docking clustering threshold*. We previously mentioned a clustering threshold for pairwise docking. Clustering is also performed against a population of protein complexes at the end of every GA generation to remove redundant models. The RMSD between C-α atoms of models is also used in this case, and it is set by the --cluster flag. The default value is set to 10 Å. By default, the cutoffs of both pairwise and multiple docking are set to 10 Å. We would recommend users to keep it at 10 Å, but they do not need to be the same value.
9. *Generation of poses in the PDB format*. After Multi-LZerD finishes, a file with a *.ga.out* suffix will be generated. It will contain the same number of models as the number provided for the --population parameter. To generate PDB files from the output file, *run.sh* executes the following command:

```
./multilzerd_create_pdb sample.ga.out ./1 200 decoy
```

   This will create 200 PDB files that start with the "decoy-" prefix, followed by a number that shows the model rank. This program receives as input five parameters:

   - A *.ga.out* output file.
   - The directory where the pairwise output files and the original PDB files are located. In the example provided, this means that *A-sample.pdb*, *B-sample.pdb*, *C-sample.pdb*, *A-B.out*, *A-C.out*, and *B-C.out* are in the current directory "./".
   - Two index numbers between 1 and the population size used that determine the rank of the PDB models to output. In the example, since the population size used was 200, 1 and 200 imply that all the models will be generated. An alternative range such as 25 to 50 means that the first 24 models will not be generated and models between 25 and 50 (inclusive) will.
   - A prefix that will be used in all of the output file names (*decoy* in the example).

10. *Refinement*. It is possible to apply a refinement program in the package to the generated models. The refinement program will perform small translations and rotations between subunits in a complex model to find a complex structure of lower energy.

Resulting models are not expected to deviate significantly from the starting conformation; rather, the program is designed for minor adjustments. This is achieved by the last command in *run.sh*:

```
./multilzerd_refine --input sample.ga.out
--trials 200 --prefix refined
```

The refinement tries a configurable number of randomized modifications, specified by the *--trials* flag. While we provide 200 as a default value, we have used 2,000 in some of our previous works. One refined model will be created for each model in the *.ga.out* file. Refined PDB files have the following naming convention: refined- < ##### > .pdb (with a five-digit number that corresponds to the prediction number in the *.ga.out* file).

4 Case Studies

Here we show examples of predicted protein complexes by LZerD, PI-LZerD, and Multi-LZerD. Figure 6a is a pairwise docking prediction by LZerD for a protein complex of a bovine beta-trypsin and an inhibitor CMTI-I (PDB ID: 1PPE). This is one of the entries in a protein–protein docking benchmark set [30]. The results for a sample LZerD [11] run are visualized with PyMOL [31] in part A of this figure. Two predicted positions for the ligand, CMTI-I, are shown. The structure in blue corresponds to *ligand1.pdb* (iRMSD 14.3 Å), while the best prediction, ranked 505 in this sample run, is shown in orange (iRMSD 1.12 Å).

The second example (Fig. 6b) is a prediction by PI-LZerD for a complex of human CDK2 kinase with cell cycle regulatory protein CksHs1. The correct docking pose is shown in blue, and the prediction by PI-LZerD is shown in green (iRMSD: 1.03 Å). The interface residue prediction used has a sensitivity (fraction of correctly predicted interface residues) of 0.33 for the receptor and 0.44 for the ligand. Interface residue information can also be used as post-screening for docking models, where models that have the same set of interface residues as the provided interface residues are selected among all of the produced docking models. The post-screening (called naïve-filtering method) did not work well for this case because the interface residue information is not very accurate (predicted pose shown in red).

Finally, we show a prediction example of Multi-LZerD (Fig. 6c). We have used a trimeric complex of two transcription factors and a retinoblastoma-associated protein (PDB ID: 2AZE). This trimer has a triangular topology, which means that all three units interact with the other two. A near-native prediction (a global C-α RMSD of 0.99 Å) was found as the lowest energy model as

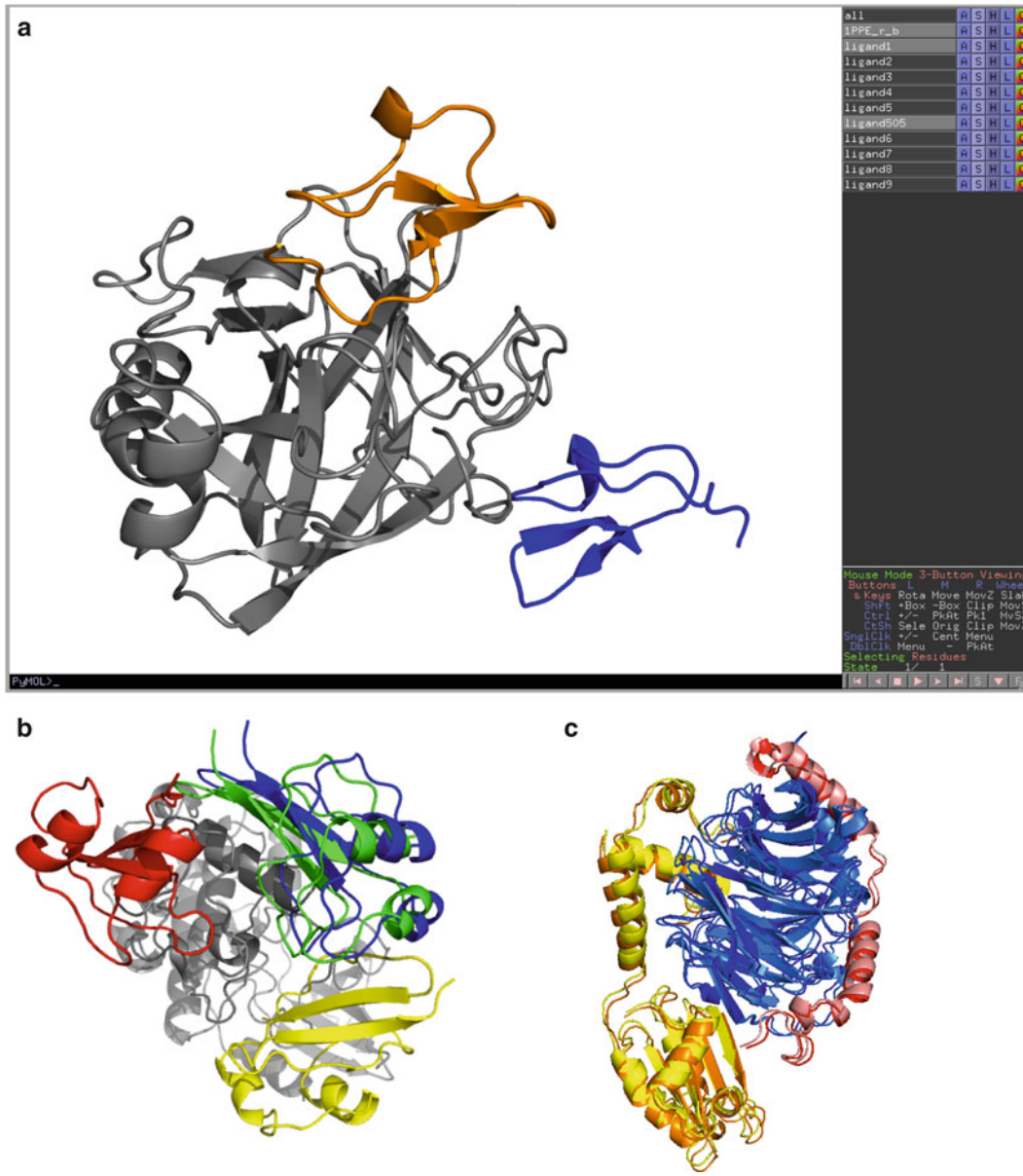

Fig. 6 Visualization of LZerD case studies. (**a**) Snapshot of visualization of predicted docking models for a protein complex (PDB ID: 1PPE) using PyMOL. Prediction 1 (*blue*) and 99 (*orange*) are visualized, while the other models are in the environment but not shown. (**b**) Predictions generated by PI-LZerD for 1BUH. The native structure is shown in *blue*, PI-LZerD's prediction in *green*, the standard LZerD in *yellow*, and the *red one* shows a prediction using the naïve-filtering method, which simply selects models by examining the consistency of interface residues in models with predicted interface residues. This figure is modified from a figure originally published in the PI-LZerD paper [23]. (**c**) A Multi-LZerD prediction for 1A0R. *Blue*, *red*, and *yellow* show the native conformation, while *light blue*, *salmon*, and *orange* show the predicted unit poses

shown in the original Multi-LZerD publication [17]. The native units are shown in blue, red, and yellow colors, while the predicted units are colored using light blue, salmon, and orange.

5 Notes

In this section we provide a few hints that we have found useful when executing our docking protocols.

1. *Running LZerD and Multi-LZerD with nohup*: We encourage users to execute either LZerD or Multi-LZerD using nohup, a Linux command that allows processes to keep running even if users log out from a computer. This is useful especially when users remotely log into a computer because it allows users to start a docking process and log out once the command is issued without having to wait for the whole process to finish.

   In addition, we suggest that the terminal output be redirected to a file, which can serve as a log of the execution. For example, to run LZerD, the user can execute this command:

```
nohup ./runlzerd.sh example/1PPE_r_b.pdb
example/1PPE_l_b.pdb >& log.txt &
```

   The ">&" tells the command shell to redirect all output, which are normally shown on the terminal screen to *log.txt*. Notice that not all command shells use the same nomenclature to redirect the output. The final ampersand symbol tells the terminal to let the program run in the background.

2. *Optimizing GETPOINTS cutoff*: While discussing LZerD we mentioned that the *-cut* parameter given to GETPOINTS could vary the runtime significantly. We suggest that users, when testing a new protein complex, first set this parameter to 1e-02 and observe how much time it takes to run the protocol. Then run the program again with 1e-03 and finally with 1e-04. The first setting is expected to have a runtime between minutes to around one to two hours. The latter two settings take more time. If the available time allows it, users can run the procedure again with the second or the third setting. If the new setting is taking longer than users can afford, the new run can be aborted, and the results from the previous run can be used.

Acknowledgments

The authors thank Kristen Johnson for proofreading the manuscript. This work has been supported by grants from the National Institutes of Health (R01GM075004 and R01GM097528), National Science Foundation (EF0850009, DBI1262189, IOS1127027, IIS1319551), and National Research Foundation of Korea Grant funded by the Korean Government (NRF-2011-220-C00004). J.E.R. would like to thank the Fulbright Science and Technology program for supporting his first years of graduate studies.

References

1. Rose PW, Bi C, Bluhm WF et al (2013) The RCSB Protein Data Bank: new resources for research and education. Nucleic Acids Res 41:D475–D482. doi:10.1093/nar/gks1200
2. Ben-Zeev E, Eisenstein M (2003) Weighted geometric docking: incorporating external information in the rotation-translation scan. Proteins 52:24–27. doi:10.1002/prot.10391
3. Chen R, Li L, Weng Z (2003) ZDOCK: an initial-stage protein-docking algorithm. Proteins 52:80–87. doi:10.1002/prot.10389
4. Dominguez C, Boelens R, Bonvin AMJJ (2003) HADDOCK: a protein-protein docking approach based on biochemical or biophysical information. J Am Chem Soc 125:1731–1737. doi:10.1021/ja026939x
5. Gray JJ, Moughon S, Wang C et al (2003) Protein-protein docking with simultaneous optimization of rigid-body displacement and side-chain conformations. J Mol Biol 331:281–299
6. Moreira IS, Fernandes PA, Ramos MJ (2010) Protein-protein docking dealing with the unknown. J Comput Chem 31:317–342. doi:10.1002/jcc.21276
7. Pierce B, Weng Z (2007) ZRANK: reranking protein docking predictions with an optimized energy function. Proteins 67:1078–1086. doi:10.1002/prot.21373
8. Ritchie DW (2008) Recent progress and future directions in protein-protein docking. Curr Protein Pept Sci 9:1–15
9. Schneidman-Duhovny D, Inbar Y, Nussinov R, Wolfson HJ (2005) PatchDock and SymmDock: servers for rigid and symmetric docking. Nucleic Acids Res 33:W363–W367. doi:10.1093/nar/gki481
10. Tovchigrechko A, Vakser IA (2006) GRAMM-X public web server for protein-protein docking. Nucleic Acids Res 34:W310–W314. doi:10.1093/nar/gkl206
11. Venkatraman V, Yang YD, Sael L, Kihara D (2009) Protein-protein docking using region-based 3D Zernike descriptors. BMC Bioinforma 10:407. doi:10.1186/1471-2105-10-407
12. André I, Bradley P, Wang C, Baker D (2007) Prediction of the structure of symmetrical protein assemblies. Proc Natl Acad Sci USA 104:17656–17661. doi:10.1073/pnas.0702626104
13. Inbar Y, Benyamini H, Nussinov R, Wolfson HJ (2005) Prediction of multimolecular assemblies by multiple docking. J Mol Biol 349:435–447. doi:10.1016/j.jmb.2005.03.039
14. Berchanski A, Eisenstein M (2003) Construction of molecular assemblies via docking: modeling of tetramers with D2 symmetry. Proteins 53:817–829. doi:10.1002/prot.10480
15. Comeau SR, Camacho CJ (2005) Predicting oligomeric assemblies: N-mers a primer. J Struct Biol 150:233–244. doi:10.1016/j.jsb.2005.03.006
16. Karaca E, Melquiond ASJ, De Vries SJ et al (2010) Building macromolecular assemblies by information-driven docking: introducing the HADDOCK multi-body docking server. Mol Cell Proteomics 9:1784–1794. doi:10.1074/mcp.M000051-MCP201
17. Esquivel-Rodríguez J, Yang YD, Kihara D (2012) Multi-LZerD: multiple protein docking for asymmetric complexes. Proteins 7:1818–1833. doi:10.1002/prot.24079
18. Wolfson HJ, Rigoutsos I (1997) Geometric hashing: an overview. IEEE Comput Sci Eng 4:10–21. doi:10.1109/99.641604
19. Canterakis N (1999) 3D Zernike moments and Zernike affine invariants for 3d image analysis and recognition. 11th scandinavian conference on image analysis
20. Novotni M, Klein R (2003) 3D zernike descriptors for content based shape retrieval. Proceedings of the eighth ACM symposium on solid modeling and applications—SM'03. ACM Press, New York, NY, USA, p 216
21. Kihara D, Sael L, Chikhi R, Esquivel-Rodríguez J (2011) Molecular surface representation using 3D Zernike descriptors for protein shape comparison and docking. Curr Protein Pept Sci 12:520–530, doi: http://dx.doi.org/10.2174/138920311796957612
22. Sael L, Kihara D (2009) Protein surface representation and comparison: new approaches in structural proteomics. In: Chen JY, Lonardi S (eds) Biological data mining. Chapman & Hall/CRC, Boca Raton, FL, pp 89–109
23. Li B, Kihara D (2012) Protein docking prediction using predicted protein-protein interface. BMC Bioinforma 13:7. doi:10.1186/1471-2105-13-7
24. Esquivel-Rodríguez J, Kihara D (2012) Effect of conformation sampling strategies in genetic algorithm for multiple protein docking. BMC Proc 6 Suppl 7:S4. doi: 10.1186/1753-6561-6-S7-S4
25. Esquivel-Rodríguez J, Kihara D (2012) Fitting multimeric protein complexes into electron microscopy maps using 3D Zernike descrip-

tors. J Phys Chem B 23:6854–6861. doi:10.1021/jp212612t

26. Esquivel-Rodríguez J, Kihara D (2012) Evaluation of multiple protein docking structures using correctly predicted pairwise subunits. BMC Bioinforma 13:S6. doi:10.1186/1471-2105-13-S2-S6
27. La D, Kihara D (2012) A novel method for protein-protein interaction site prediction using phylogenetic substitution models. Proteins 80:126–141. doi:10.1002/prot.23169
28. Qin S, Zhou H-X (2007) meta-PPISP: a meta web server for protein-protein interaction site prediction. Bioinformatics 23:3386–3387. doi:10.1093/bioinformatics/btm434
29. McDonald IK, Thornton JM (1994) Satisfying hydrogen bonding potential in proteins. J Mol Biol 238:777–793. doi:10.1006/jmbi.1994.1334
30. Hwang H, Vreven T, Janin J, Weng Z (2010) Protein-protein docking benchmark version 4.0. Proteins 78:3111–3114. doi:10.1002/prot.22830
31. Schrödinger L (2010) The PyMOL Molecular Graphics System, Version 1.5.0.4