LightDock 4G6M example

This is a complete example of the LightDock protocol when residue restraints are specified using the 4G6M complex as an example.

All the files used in this example can be found in the path examples/4G6M.

IMPORTANT: We recommend you to create a new folder and to copy the starting files `4G6M_rec.pdb`, `4G6M_lig.pdb` and `restraints.list` to that folder in case you would like to run this example.

1. Setup

First, run the setup:

```
lightdock_setup 4G6M_rec.pdb 4G6M_lig.pdb 400 200 -anm --noxt -rst restraints.list
```

The output should be something like this:

```
@> ProDy is configured: verbosity='info'
[lightdock_setup] INFO: Reading structure from 4G6M_rec.pdb PDB file...
[lightdock_setup] INFO: 1782 atoms, 230 residues read.
[lightdock_setup] INFO: Reading structure from 4G6M_lig.pdb PDB file...
[lightdock_setup] INFO: 1194 atoms, 149 residues read.
[lightdock_setup] INFO: Calculating reference points for receptor 4G6M_rec.pdb...
[lightdock_setup] INFO: Done.
[lightdock_setup] INFO: Calculating reference points for ligand 4G6M_lig.pdb...
[lightdock_setup] INFO: Done.
[lightdock_setup] INFO: Saving processed structure to PDB file...
[lightdock_setup] INFO: Done.
[lightdock_setup] INFO: Saving processed structure to PDB file...
[lightdock_setup] INFO: Done.
[lightdock_setup] INFO: 10 normal modes calculated
[lightdock_setup] INFO: 10 normal modes calculated
[lightdock_setup] INFO: Reading restraints from restraints.list
[lightdock_setup] INFO: Number of receptor restraints is: 20 (active), 0 (passive)
[lightdock_setup] INFO: Number of ligand restraints is: 21 (active), 0 (passive)
[lightdock_setup] INFO: Calculating starting positions...
[lightdock_setup] INFO: Generated 84 positions files
[lightdock_setup] INFO: Done.
[lightdock_setup] INFO: Number of swarms is 84 after applying restraints
[lightdock_setup] INFO: Preparing environment
[lightdock_setup] INFO: Done.
[lightdock_setup] INFO: LightDock setup OK
```

2. Simulation

We can run our simulation in a local machine or in a HPC cluster. For the first option, simply run the following command:

```
lightdock setup.json 100 -s fastdfire -c 8
```

Where the flag `-c 8` indicates LightDock to use 8 available cores. For this example we will run `100` steps of the protocol and the C implementation of the DFIRE function `-s fastdfire`.

To run a LightDock job on a HPC cluster, a Portable Batch System (PBS) file can be generated. This PBS file defines the commands and cluster resources used for the job. A PBS file is a plain-text file that can be easily edited with any UNIX editor. For example, create a `submit_job.sh` file containing:

```
#PBS -N LightDock-4G6M
#PBS -q medium
#PBS -l nodes=1:ppn=16
#PBS -S /bin/bash
#PBS -d ./
#PBS -e ./lightdock.err
#PBS -o ./lightdock.out

lightdock setup.json 100 -s fastdfire -c 16
```

This script tells the PBS queue manager to use 16 cores of a single node in a queue with name `medium`, with job name `LigthDock-4G6M` and with standard output to `lightdock.out` and error output redirected to `lightdock.err`.

To run this script you can do it so:

```
qsub < submit_job.sh
```

3. Analysis

Once the simulation has finished (it takes around 1-2 min per 10 steps per swarm), you should:

1. Generate structures per swarm (200 glowworms per swarm in this example)
2. Cluster predictions per swarm
3. Generate the ranking files
4. Filter by a percentage of satisfied restraints (>40% in this example)

Here there is a PBS script to do so:

```
#PBS -N 4G6M-anal
#PBS -q medium
#PBS -l nodes=1:ppn=8
#PBS -S /bin/bash
#PBS -d ./
#PBS -e ./analysis.err
#PBS -o ./analysis.out

### Calculate the number of swarms ###

s=`ls -d ./swarm_* | wc -l`
swarms=$((s-1))


### Create files for Ant-Thony ###

for i in $(seq 0 $swarms)
  do
    echo "cd swarm_${i}; lgd_generate_conformations.py ../4G6M_rec.pdb ../4G6M_lig
.pdb  gso_100.out 200 > /dev/null 2> /dev/null;" >> generate_lightdock.list;
  done

for i in $(seq 0 $swarms)
  do
    echo "cd swarm_${i}; lgd_cluster_bsas.py gso_100.out > /dev/null 2> /dev/null;
" >> cluster_lightdock.list;
  done


### Generate LightDock models ###

ant_thony.py -c 8 generate_lightdock.list;


### Clustering BSAS (rmsd) within swarm ###

ant_thony.py -c 8 cluster_lightdock.list;


### Generate ranking files for filtering ###

lgd_rank.py $s 100;


### Filtering models by >40% of satisfied restraints ###

lgd_filter_restraints.py --cutoff 5.0 --fnat 0.4 rank_by_scoring.list restraints.l
ist A B > /dev/null 2> /dev/null;
```

When the analysis is finished, a new folder called `filtered` has been created, which contains any predicted structure which satisfies our 40% filter and a file with the ranking of these structures by LightDock DFIRE (fastdfire) energy (more positive better) `rank_filtered.list`.

We provide for this example a compressed filtered folder [filtered.tgz] which contains (when decompressed) a ranking `lgd_filtered_rank.list` file. For all the filtered structures, interface RMSD (i-RMSD), ligand RMSD (l-RMSD) and fraction of native contacts (fnc) according to CAPRI criteria has been calculated:

```
$ head lgd_filtered_rank.list
# structure    i-RMSD  l-RMSD  fnc      Score
swarm_83_154.pdb    2.091   2.012   0.603448    56.869
swarm_50_151.pdb    3.082   4.001   0.327586    50.536
swarm_7_192.pdb 1.553   1.461   0.586207    48.936
swarm_36_168.pdb    1.132   1.385   0.827586    48.870
swarm_48_19.pdb 3.687   4.205   0.258621    48.739
swarm_11_136.pdb    2.231   2.390   0.448276    48.662
swarm_52_171.pdb    3.933   4.052   0.155172    47.808
swarm_49_183.pdb    3.589   3.775   0.258621    47.659
swarm_48_49.pdb 1.562   2.100   0.793103    47.234
swarm_65_93.pdb 1.282   1.130   0.844828    45.372
```

As you may observe, for this example our protocol seems to perform extremely well when restraints close to the true interface are specified.