# Manual of EModel2D

Last uptade: April 2012
Author: Javier Velazquez-Muriel

## 1) Introduction

EModel2D is an application to build models of macromolecular assemblies using restraints from EM images. It is based on the paper:

xxxxxxxxxxxxxx

Apart from EM images, the method uses some other restraints:

- Connectivity restraints enforcing a set of components of the assembly to be within a certain distance, suitable for proteomics data.
- Maximum distance restraints imposing a maximum distance between a pair of residues, as derived from cross-linking experiments.
- Geometric complementarity restraints favoring large contact surfaces between interacting components.
- Excluded volume restraints preventing two components of the assembly from interpenetrating.

EModel2D samples for good conformations of the macromolecular assembly using

- Multiple molecular docking.
- Simulated annealing Monte Carlo optimization.
- Sampling with the discrete optimizer DOMINO.

It is straightforward to incorporate other restraints to the method.

## 2) Installation and requirements

EModel2D is part of IMP, and therefore has all the dependencies that IMP has. But you are aware of it if you are reading this. On top of them, the em2d module, which is the base of this application, has some other requirements. The list is intimidating, but you'll see that it makes sense:

- **Python2.7.** It is probably on your machine already.

- **openCV.** openCV is a library for computer vision that provides a lot of functionality for dealing with images. EModel2D uses it for speed. We have tested that the application works with OpenCV 2.1, 2.2 and 2.3. Installing openCV should not be very traumatic. Here is are some quick instructions that work for us:

On linux:

- Download the source code from http://opencv.itseez.com/ and put it in a directory, `opencv_install` for example. Go to this directory.
- Call `cmake` to build the library. There is only one thing to pay attention to, the directory where you want to install the program:

```
cmake  -D  CMAKE_BUILD_TYPE=RELEASE  -D  CMAKE_INSTALL_PREFIX=/home/you/OpenCV-2.3.0/                 -D BUILD_PYTHON_SUPPORT=ON ..
```

- In our case there was one more trick. We had to go to the file `MakeCache.txt` and change `PYTHON_EXECUTABLE:FILEPATH=/usr/bin/python2.3` for `PYTHON_EXECUTABLE:FILEPATH=/usr/bin/python2.7`
- Execute `make`
- Execute `make install`
- Now you should have openCV in the directory `/home/you/OpenCV-2.3.0/`. You may have some other problems, and here is the point where you will want to go to the openCV documentation and fight a bit with it.

On Mac:
MacPorts, a nice tool to get Unix programs working on Mac, provides a version of openCV. Installing openCV should be really easy. Learn a bit about the "port" command from macports, and port openCV. It should be only a line.

```
sudo port install opencv
```

Remember that as any other dynamic library, you must have its path in the environment variable `LD_LIBRARY_PATH`. In our example, the path to include is `/home/you/OpenCV-2.3.0/lib`.

- The GNU scientific library http://www.gnu.org/software/gsl/. EModel2D uses it because it has a good implementation of the Simplex method for function optimization. If you have an standard linux box, most probably it is already installed. The gsl module in IMP should then take care of the rest. For mac, as simple as before:
  ```
  sudo port install gsl
  ```
- The docking program HEXDOCK. http://hex.loria.fr/. The program is optional but very good to have. It is not required for building the em2d module or IMP.

If you have all this dependencies, IMP will compile the em2d module. Of course the em2d module depends on other modules: core, atom, em, gsl, container, em. But you already have them.

## 3) Input data

You need only three things to do a modeling with EModel2D:

1. A set of PDB files with the components of your assembly.
2. A set of EM images.
3. A configuration file.

Let's describe them a little bit.

**The PDB files with the components of the assembly**. Each PDB file must contain a protein or DNA strand in a chain. The chains must have different ID. i.e., do not use chain A for two different components. The atoms in the chains are the atoms that you want to use. If you don't want to have duplicated atoms, due to PDB records as ANISOU, you need to remove them. We tend to be more radical and remove other records like REMARK, SOURCE, COMPND, SEQRES, DBREF, CONECT. They are no relevant for the type of problem that EModel2D solves.

**The EM images**. IMP can understand 3 image formats. Spider, JPG and TIFF. The format that you want to use 99% of the time is Spider, as it is specific for EM. Of course there are other formats for EM, and you can use the free program `em2em` to convert them to Spider. Conversion to JPG and TIFF are nice to have as more traditional graphic formats. There is a script in the directory of this manual to do conversions.
Each EM image has to be a separate file. The images that you want to use need to be listed in a "selection file". This is just a file with 2 columns, with the name of the images and 0/1. If there is a 0, that image is not used. For example, a selection file `myselection.sel` like this:
image1.spi 1
image2.spi 0
image3.spi 1
means that you have 3 images but only want to use image1 and image3 for the modeling.

**A configuration file**. A configuration file is just a python file with classes that describe all the parameters and restraints that you want to use. Using a python file as configuration file makes adding new parameters to your simulation trivial. It is better to describe the file it in situ, so open the file `config_example.py` located in the example_3sfd directory to find a detailed description of each of the parameters. The file is easier to read if you have an editor with syntax highlighting for python. Once that you understand all options, you can of course remove all the lengthy comments.

## 4) How to get models

Getting models requires 4 steps:
1. Doing the preliminary dockings.
2. Obtaining models with Monte Carlo optimizations.
3. Gathering the solutions from the Monte Carlo optimizations.
4. Combining models from Monte Carlo with DOMINO to get even better models.

### Preliminary dockings
EModel2D performs docking between components that are subject to cross-linking restraints. EModel2D uses the program HEXDOCK by Ritchie et. al, 2010. This is what EModel2D does:
- Finds a very rough estimation of the orientation between two components by minimizing the distance between the aminoacids implied in the cross-linking restraints. Admittedly, this is going to be really bad, but there is a trick. It helps the HEXDOCK program providing a "hint" of the orientation. You then tell HEXDOCK that you don't want to search all possible orientations for the ligand, just a given angle around the orientation obtained from the rough guess. In our experience, it works well in many cases.

- Determines an optimal way of doing the required dockings. It finds the component of the assembly that needs to be kept anchored, and establish an order for the dockings. The docking order is based on the maximum spanning tree of the graph built by the component connections. Here is an example: Let's say that you have a complex with 5 chains: A B C D E, and your cross-linking restraints say that the components should be connected like this:
  ```
   A-B-D-E
     \ | /
       C
  ```
  The weight for an edge is the number of restraints between the components that it connects. After computing the maximum spanning tree, you could get this graph:
  ```
  A-B-D-E
    |
    C
  ```
  Which says that B should be anchored and be the first receptor, as it is the component with the largest number of neighbors. And edge indicates that a docking should be done. In our case:
  A (ligand) docked to B (receptor)
  C docked to B
  D docked to B
  E docked to D
- Computes the dockings using HEXDOCK. Optionally, you can get the docking order computed in the previous step and use your favorite docking program/server to do these dockings. You will need to recover from your docking program the relative transformation of the ligand respect to the receptor for each solution. See the explanation for the `em2d_docking.py` script to see how you could integrate your favorite program.
- Filters the docking solutions that are compatible with the cross-linking restraints. As an emergency measure, if there are no solutions compatible with the restraints, all of them are taken. Of course this implies the risk of using solutions that are not very accurate.

The entire procedure described above can be done with the command:

```
imppy.sh python em2d_domino_model.py --exp config.py --dock --log file.log
```

Now you have to take the information from the dockings and put it in the configuration file. You need to indicate which component is anchored and provide the files of relative transformations from the dockings. The options to modify are `self.anchor` and `self.dock_transforms`. The example of section 6 will show you how.

*NOTE:* The option `--log` is not mandatory. If you use it (recommended), you will get a file with the information for the modeling. Otherwise, all the information will be printed on the screen. This logging information is coming only from the python interface of this application, and is different from the IMP logging system.

**Obtain models with Simulated annealing Monte Carlo optimization.**

Once that the relative docking transformations are set, you have to do Monte Carlo optimizations for getting models. The command is:

```
imppy.sh python em2d_domino_model.py --exp config.py --o montecarlo1.db --log
file.log --monte_carlo -1
```

The -1 for the monte_carlo option is explained in the help for `em2d_domino_model.py`. What you get after the optimization is a SQLite database `(montecarlo1.db` in this example) with only one solution. This is so because the idea is to run `domino_model.py` with Monte Carlo as many times as models you want. In a computer cluster all this can be done in parallel.

**Gather the results of all Monte Carlo optimizations.**
Now is time to put all the Monte Carlo solutions together. It is done with the command:

```
imppy.sh python em2d_domino_model.py --o all_montecarlo.db --gather {all database
files}
```

Where {all database files} means the name of all the files to join. Something like montecarlo*db, if you decided to use `montecarlo1.pdb, montecarlo2.pdb,` etc. as the names of the databases.

**Combine the models from Monte Carlo with DOMINO**
The solutions in `all_montecarlo.pdb` are already solutions for the modeling. They are a set of discrete solutions that can be improved by combining the positions of the components in all of them. For example, if you have 100 solutions from the Monte Carlo experiments, then you have 100 possible positions for each component. The positions should (hopefully) be already quite correct, but what you can achieve with DOMINO is to explore all the possible combinations. If the assembly that you want to model has 4 components, using DOMINO you are exploring the 100^4 possible combinations. The models found will be better than the Monte Carlo ones. To do the task, first you have to go the configuration file and change the value of the member variable `self.read` in the `DominoSamplingPositions` class. The value that you want is the name of the database of Monte Carlo results. For example, `all_montecarlo.db`. Then run the command:

```
imppy.sh python em2d_domino_model.py --exp config.py --o domino_models.db --log
file.log
```

This will produce a database `domino_models.db` with all the results.

## 5) Visualizing the models and understanding the information in the database of solutions

The database of results contains all the positions for the rigid bodies in the solutions. To write some of these solutions, the command is:

```
imppy.sh python em2d_domino_model.py --exp config.py --o domino_models.db --w 10 --
orderby em2d --log file.log
```

In this case the option `--o` does not modify the database, only uses it for reading the positions of the components. The option `--w` says that we want 10 models. The option `--orderby` is the name of the restraint used to sort the models. When using `--orderby em2d` in the example above, you say that you want the 10 best models according to the value of the em2d restraint. Another typical option is total_score. The solutions are written to the files `solution-000.pdb, solution-001.pdb`, and so on. Each record for a solution in the database contains the following information:

1. **Solution_id** - A unique number that identifies the model. Note: solution_id=0 does not mean the best solution. It is only an identifier.
2. **assignment** - Is the set of numbers identifying a combination in domino. For the previous example with 4 components an 100 positions, one assignment could be "11|23|45|76".
3. **Reference frames**. These are the values used to build an `algebra.ReferenceFrame3D` object in IMP. There is one reference frame per component of the assembly. Generating a solution is as simple as setting the reference frame of each of the rigid bodies of the components of the assembly.
4. **Total_score** - The total value of the scoring function.
5. **{restraints}** this is a list of values for the restraints. There is one column in the database for each restraint. The list changes with the number and nature of the restraints.

Using a database output file is very powerful, as you can query the data and decide how you want the information. On the other side, it is less comfortable than a simple text file. To help with that, see the file `quick_sql_query.py` in the directory of the example, which contains some typical SQL queries.

## 6) A complete example

Here is an entire example for one of the experiments in the paper. It is the modelling of the structure with PDB ID 3sfd. You can find it in the subdirectory `example_3sfd`.

- The inputs are the files 3sfdA.pdb, 3sfdB.pdb, 3sfdC.pdb, 3sfdD.pdb and the images in the directory em_images. The selection file is `images.sel`.
- To do the dockings, run

```
imppy.sh python em2d_domino_model.py --exp config_step_1.py --dock --log dock.log
```

  Open the file `dock.log` and search for the line:

```
INFO:buildxlinks:The suggested order for the docking pairs is [('3sfdB', '3sfdA'), ('3sfdB', '3sfdC'), ('3sfdB', '3sfdD'), ('3sfdD', '3sfdC')]
```

  This line is telling you the dockings required, and the order recommended. The pairs are (Receptor,Ligand), so you need to dock 3sfdA into the 3sfdB, 3sfdC into 3sfdB, 3sfdD into 3sfdB, and 3sfdC into 3sfdD. If you have HEXDOCK and everything went well, you'll see a lot of new files. They are in the outputs directory.
    - Files with the ligand in the position of the rough estimation of the orientations based on the cross-links:
      3sfdB-3sfdD_initial_docking.pdb
      3sfdB-3sfdA_initial_docking.pdb
      3sfdD-3sfdC_initial_docking.pdb
      3sfdB-3sfdC_initial_docking.pdb
    - Files with the first solution found by HEXDOCK:
      3sfdB-3sfdA_hexdock.pdb
      3sfdB-3sfdC_hexdock.pdb
      3sfdB-3sfdD_hexdock.pdb
      3sfdD-3sfdC_hexdock.pdb
    - Files of all the transformations of the ligand from HEXDOCK:

hex_solutions_3sfdB-3sfdA.txt
hex_solutions_3sfdB-3sfdD.txt
hex_solutions_3sfdB-3sfdC.txt
hex_solutions_3sfdD-3sfdC.txt

- Files with the filtered solutions:
hex_solutions_3sfdB-3sfdA_filtered.txt
hex_solutions_3sfdB-3sfdC_filtered.txt
hex_solutions_3sfdB-3sfdD_filtered.txt
hex_solutions_3sfdD-3sfdC_filtered.txt
- Files with the relative transformations of the ligand respect to the receptor:
relative_positions_3sfdB-3sfdA.txt
relative_positions_3sfdB-3sfdD.txt
relative_positions_3sfdB-3sfdC.txt
relative_positions_3sfdD-3sfdC.txt

The files with the relative transformations are the files that we want. The component to anchor is 3sfdB, because in the component with most neighbours. It is easy to identify because it is the first receptor in the list of docking pairs.
What happens if you don't have HEX? You can specify that in the configuration file with `self.have_hexdock = False`. You will still get the order suggested, but you have to do the dockings with your favorite program. You have to compute the relative orientations of the ligand respect to the receptor and translate them into IMP transformations. If you can do that, then obtaining a file like `relative_positions_3sfdB-3sfdA.txt` is not difficult. Assuming that you have found the relative transformation given by 3 Euler angles ZYZ (phi, theta, psi) and a translation (x,y,z), here is the set of commands in IMP that will give you the transformation:

```
R = IMP.algebra.get_rotation_from_fixed_zyz(phi, theta, psi)
q = R.get_quaternion()
```

What you see in the file relative_positions_3sfdB-3sfdA.txt is just:
q[0] | q[1] | q[2] | q[3] | x | y | z

- Once that you have the dockings and the anchored component, go to the configuration file config_step_1.py and fill the values for `self.anchor` and `self.dock_transformations`:

```
self.anchor = [False,True,False,False]
self.dock_transforms =  [
      ["3sfdB","3sfdA","relative_positions_3sfdB-3sfdA.txt"],
      ["3sfdB","3sfdC","relative_positions_3sfdB-3sfdC.txt"],
      ["3sfdB","3sfdD","relative_positions_3sfdB-3sfdD.txt"],
      ["3sfdD","3sfdC","relative_positions_3sfdD-3sfdC.txt"],
                         ]
```

The file `config_step_2.py` contains all the changes.

- The next step is running a Monte Carlo optimization. You can adjust the profile of temperatures, number of iterations, cycles, maximum displacement and angle tolerated for the random moves, and also the parameter `self.non_relative_move_prob`. This parameter indicates the probability for a component of doing a random movement instead of a relative movement. If you put 0.4, it means that the component 3sfdA will do a random move instead of moving to a relative position respect to its receptor

3sfdB 40% of the time. The same applies to all other pairs of components. To run the optimization:

```
imppy.sh python em2d_domino_model.py --exp config_step_2.py --monte_carlo -
1 --log monte_carlo.log --o mc_solution1.db
```

Probably you noticed that there were changes in the MonteCarloParams of `config_step_2.py` respect to `config_step_1.py.` The new parameters for Monte Carlo were set to get a very very short simulation. You'll get a garbage model (quickly). The actual parameters used during the benchmark for the paper are those on `config_step_1.py`. Once the script has finished, there should be two new files in the directory, the logging file and the database with the result:

monte_carlo.log
mc_solution1.db

- You can do many Monte Carlo modelings to obtain files like mc_solution1.db and gather them with the command:
  ```
  imppy.sh python em2d_domino_model.py --o monte_carlo_solutions.db --gather
  mc_solution*.pdb
  ```
  For this example we have included the file `monte_carlo_solutions.db`, which contains the results of 500 Monte Carlo modelings.

- The last part of the modeling is running DOMINO employing the config file `config_step_3.py`. `config_step_3.py` has changes respect to `config_step_2.py` in the classes DominoSamplingPositions and DominoParams. The parameters are:
  - **self.read** is the file with the Monte Carlo solutions obtained before.
  - **self.max_number** is the maximum number of solutions to combine. In this example, with 500 solutions and 4 components, we would have to explore 500^4 combinations. This number allows you to reduce that. Fore the example, we set 5, and therefore only 5^4 combinations are explored.
  - **self.orderby** is the name of the restraint used to sort the Monte Carlo solutions. Here the value is "em2d", so the best 5 solutions according to the em2d score are combined with DOMINO. You could try using "total_score" too.
  - **self.heap_solutions.** This is a rather technical parameter. It is the number of solutions that you keep each merging step in DOMINO. The larger the number, the better the space of 5^4 combinations is explored, at the cost of larger running time. Here we put 200.

  The command is:

```
imppy.sh python domino_model.py --exp config_step_3.py --log domino.log --o
domino.db
```

You will get files domino.log and domino.db, containing the logging and the database of solutions, respectively. Once again, the parameters used for domino were selected to get a quick answer. For the example, we have included the file domino_solutions.db, which we obtained during our benchmark. We used `self.max_number=50` and `self.heap_solutions=2000.`

- To write some solutions run:
  ```
  imppy.sh python em2d_domino_model.py --exp config_step_3.py --w 10 --o
  domino.db --orderby em2d
  ```

The solutions will be in the files `solution-*.pdb`. They will not be very good, because they came from sampling only 5^4 combinations. But try the file domino_solutions.db that we obtained during our benchmark:

```
imppy.sh python em2d_domino_model.py --exp config_step_3.py --w 10 --o
domino_solutions.db --orderby m2d
```

- Finally, you could be interested in querying the values of the restraints for the models in the database file domino_solutions.db. In this case the file also contains measures about the quality of the models because it comes from a benchmark. Some examples of how to query the database of results are in the script `quick_sql_query.py`.

## 8) Individual description of the scripts

- **em2d_domino_model.py**. As you already learnt, this script can be used for all the stages of modeling: Docking, Monte Carlo sampling, gathering of solutions from the Monte Carlo runs, DOMINO sampling, and finally write the resulting models.
- **em2d_docking.py** - A wrapper for the program HEXDOCK. It uses HEXDOCK in text mode to perform a docking  of a subunit (the ligand) into another subunit (the receptor). The script can be used as a standalone program to perform a docking or write the solutions. The script can be modified to use any docking program, by doing a couple of changes:
    - The class `HexDocking` is called only from `em2d_omino_model.py` and uses the `dock()` method. You can chage HexDocking for another wrapper class providing a `dock()` method that saves the results to a file fn_transforms.
    - domino_model.py also calls the functions `read_hex_transforms()` and `filter_docking_results()`. You only need to adapt the function `parse_hex_transform()`, which both of them use, to your docking program.
- **em2d_cluster solutions.py**. Performs clustering of the solutions stored in a database file. You can run it as standalone program. The help of the script gives the parameters required, and a typical command is:

```
imppy.sh python em2d_cluster_solutions --exp config.py --db
domino_solutions.db --o clusters.db --n 100 --orderby em2d --log clusters.log
--rmsd 10
```

- **em2d_score_updated.py**. This script computes the em2d score for a model using the EM images. It  is useful for comparing models obtained by other sampling algorithms apart from the one described in the paper. This script supersedes the scripts em2d_score and em2d_single_score that you will find in the `build/bin` directory. To score a model the parameters required are:
    - The PDB file of the model.
    - The selection file for the EM images.
    - The pixel size of the EM images
    - The number of projections used for the coarse registration step of the scoring.
    - The resolution used to generate the projections. The model is downsampled to this value of the resolution before projecting it.
    - Images per batch. This parameter is used to avoid running out of memory when the number of images used for scoring a model is large. The scoring is done keeping in memory only the number of images specified by the parameter.

  An example:

```
imppy.sh python em2d_score_model_updated structure.pdb myimages.sel 3.6 20 5 100
```

- **convert_spider_to_jpg.py**. Does what it says.

Other scripts can be found in the directory `pyext/src` of the em2d module.

- **buildxlinks.py** - Contains all the code for generating the order of the dockings. It also contains the class InitialDockingFromXlinks, which is used to move the position of the subunits acting as ligand close to the receptor. The movement minimizes the distance between the aminoacids subject to cross-linking restraints.
- **DominoModel.py**.Contains the `DominoModel` class, which has a IMP.Model as the main member. The class manages the details of setting the model restraints, performing the Monte Carlo runs, configuring the DOMINO sampler, and storing the results in a database.
- **MonteCarloRelativeMoves.py**. Contains the class `MonteCarloRelativeMoves` for setting and configuring a simulated annealing Monte Carlo optimizer. It also manages the profiles of temperature and iterations for the sampling. The optimizer uses one `em2d.RelativePositionMover` object per docking to propose relative moves of a ligand respect to the receptor.
- **restraints.py**. Creates the restraints used for the modeling. It is called from `DominoModel.py.`
- **sampling.py**. The script allows you to set the positions and orientations for the components of the assembly before combining them using DOMINO. In the paper we ended using the set of Monte Carlo solutions, but you can use the script to set any other combination of positions and orientations for the the subunits.
- **solutions_io.py**. Contains the class `ResultsDB` for managing the database of solutions obtained during modeling.
- **Database.py, argminmax.py**, **csv_related.py,** and **utility.py** are supporting scripts. `ResultsDB` inherits all the basic functionality from `Database.py`, a wrapper for SQLite databases. The wrapper is easy to use, general, and it does not dependent on IMP, so it may be useful for managing your data too.

Some other scripts are stored in the directory `pyext/src/imp_general`. These scripts are general and perform basic and/or frequent tasks in IMP. They can be helpful for your own IMP scripts:

- **representation.py**. The main script. It contains functions for obtaining the representation of an assembly from one or more PDB files, creating rigid bodies for the components of the assembly, simplifying the structure of a protein using beads, getting coordinates and distance between residues, etc.
- **alignments.py.** A couple of functions to align assemblies.
- **comparisons.py**. Functions to compute the cross-correlation coefficient between density maps, RMSD and DRMS between models, and placement score for the subunits of an assembly as defined in the paper.
- **movement.py**. Functions for transforming a rigid body or a structure.