# Documentation of EvoDock

**Evolutionary Algorithms in Optimization of Proteins**

30.05.2021

created by

Maximilian Edich

# Table of Contents

# 1 What is EvoDock?

EvoDock is a scientific command line tool written in python, which uses an Evolutionary Algorithm (EA) to optimize ligand binding proteins. It performs automatically mutagenesis of a given protein model and refines the structure afterwards. This is followed by a ligand docking and the evaluation of the binding energy to determine the fitness of the mutational variant. The resulting output is a set of suggested optimized proteins, which can undergo further analysis in the laboratory. Alternatively, all possible mutations at certain positions can be examined to explore preferences for certain amino acid properties. Since it is designed in a modular setup, EvoDock allows a simplified implementation of modules related to other protein optimization problems.

The tool itself provides an EA and the interface to external molecular modeling software. During the initial version, only PyRosetta was implemented, but the modular setup allows an easy and standardized incorporation of other tools as well.

EvoDock was created by Maximilian Edich at the Bielefeld University in Germany as a part of his master thesis project.

# 2 What is required to run EvoDock?

EvoDock can be started with *python 3*. To make the so-called pipeline modules work, additional software must be installed on your machine. The initial version of EvoDock requires *Pyrosetta Python 3.7 Release* for proper work of the modules *MutateByPyRosetta*, *DockByPyRosetta*, and *ScoreByPyRosetta*.
Using EvoDock requires a lot of computational power, so it is preferred to use a computer cluster to benefit from parallelization of processes. The runtime of EvoDock depends highly on the input protein, specifically on its size and complexity. It depends also on the set parameters for the evolution.

# 3 How is EvoDock structured and how does it work?

The overall resulting workflow of EvoDock is depicted in Figure 1. The general pipeline, which handles external tools, hides behind the *calculate fitness* step of the workflow and is shown in detail in Figure 2. From now, the main module, which is everything except the pipeline modules, is referred to as *Evolution module*. While the *Evolution Module* handles the population of mutational variants from the input protein, as well as creating new mutants and removing bad results from the population, the *Pipeline* handles the fitness calculation.


**Figure 1:** ***Workflow of EvoDock.***
*The workflow consists of four major steps, the preparation, initialization, evolution, and output generation. From all four, only the preparation step is performed manually. The resulting input files are checked for compatibility and are used to create the initial population during the initialization. Based on the settings, the process of evolution is repeated several times. The exact routine is customized, but in general it consist of creating new offspring, assign fitness scores to each individual, and select individuals for the next cycle. At the end of the routine or if any set termination condition was met, the program exits and generates the final output. Additional output is generated during the fitness calculation.*


**Figure 2:** ***General pipeline model.***
*The pipeline of the scoring process, which determines the individual's fitness. The output of the mutagenesis consist of mutated variants from the input protein. The application uses this mutant variant to perform the task for which the input protein must be optimized. Finally, the results from the application and may the results from the mutagenesis are evaluated in respect to the task and the selection pressure. The resulting fitness is then sent back to the evolution module.*

# 4 How to use EvoDock?

This guide focuses on the usage of EvoDock's initial version, which includes the pipeline modules *MutateByPyRosetta*, *DockByPyRosetta*, and *ScoreByPyRosetta*. Using alternative modules might require different preparation.
The general usage of EvoDock is via the command line. Use python3 to start EvoDock.py with the flags from Table 1. The initial settings file and routine file are required and are explained in sections 4.2 and 4.3 respectively. These files manage all further settings.

**Table 1:** *Arguments of EvoDock.*

| Argument | Description |
|---|---|
| -h | show help message. |
| -s *file-path* | specify path to initial settings file. |
| -r *file-path* | specify path to routine file. |
| -o *file-path* | specify output path (optional). |

## 4.1 Prepare Structures

### 4.1.1 PDB Files

First of all, a pdb file containing a protein structure bound to a target ligand must be provided. If a monomer of the protein is capable of binding the ligand, the pdb file should only contain a monomer. If the binding site emerges only, if multiple instances of the protein bind to a complex, those must be provided. All in all, only one ligand-protein pair should be included. Multiple instances may work as well, but these were not tested so far. Also, more atoms in the macro molecule are likely to increase the overall computation time per mutational variant, especially during the refinement step.

### 4.1.2 Residue Selection

After preparing the pdb file, target residues must be selected. The strategy for selecting those can vary. It is recommended to pick only a small number of residues in the range of one to three for the first run, to examine the effects of mutations at these positions. It is helpful to first, locate all residues located in the region of the binding site or being in the range of a few $\mathring{A}$. Next, classify the residues from those for having direct interaction to the ligand or to nearby water molecules. Consider also a potential function of the residue for the given protein. You may consider also restricting the mutations to certain amino acids, if you would like to conserve a property like the hydrophobicity, for example. Use for all of these selection additional tools like PyMOL or the online RCSB PDB ligand interaction viewer.

### 4.1.3 Water Modeling

If water seems important in your specific situation, you might also consider to model either implicit or explicit water. While modeling of implicit water is controllable through PyRosetta's score functions, explicit water requires additional structure preparation. Since "O" atoms are removed from the pdb automatically by PyRosetta, water molecules of interest must be replaced by a water model of choice. Several water models are available, so check those out to find the ones, which suit best to your case.

### 4.1.4 Ligand Registration

To improve how PyRosetta handles the ligand, it can be registered. This process makes the molecule "familiar" to PyRosetta and could save some repetitive calculations. An official guide for this is found here:
http://www.pyrosetta.org/obtaining-and-preparing-ligand-pdb-files

## 4.2 Writing initial settings file

The initial settings file is a flag-file, containing the values to adjustable parameters. All specifications, like file paths, mutable residues, etc. are made here. All you have to do is creating a text file, add those specifications, and enter the path to it when starting EvoDock. Table 2 and Table 3 list all possible options. Depending on the external modules, some module-specific options are may added as well. An example for an initial settings file is found in the git repository within the "examples" folder. A shorter example is depicted in Figure 3.
Below is a description of some of all option in written text, explaining those in more detail and better understandable language.

**Table 2:** *Required specifications of the initial settings file.*

| Specification | Description |
|---|---|
| -task *mode* | Specify the type of optimization problem. Valid inputs are: 'LIGAND-BINDING'. |
| -targetscore $n$ | Specify the target score, which defines the best fitness. The meaning of the score depends on the specified task. $n$ must be either a float value or 'INFINITE' or 'NEGATIVE-INFINITE'. |
| -prot-path *file-path* | Specify the path to the input PDB. |
| -res-id *chain id* | Enter the residue chain and id of a residue of interest, that can be modified. Use this specification multiple times to specify multiple residues. |
| -substitutions *list* | Specify allowed mutations, where *list* consists of single letter AA codes separated by a comma. Giving no argument allows all 20 AAs. Use this specification multiple times, where the order is respective to the order of '-res-id'-commands. |
| -init-pop-run-mode *mode arg* | Specify how to get the initial population and how to proceed, where *mode* is either 'create-and-quit', 'load-and-evolve', or 'create-and-evolve', with *arg* as the number of individuals in the initial population or the path to an initial population in case of loading it. |
| -mutate *module* | Specify the name of the used mutagenesis module. |
| -apply *module* | Specify the name of the used application module. |
| -score *module* | Specify the name of the used evaluation module. |

The order of the specifications is with a few exceptions not relevant. For EvoDock it is important how to optimize the given protein. Relevant specifications for this are the

*-task*, which currently can only be LIGAND-BINDING, and the *-targetscore*. In the case of ligand binding, a score is calculated by the binding energy multiplied by -1, meaning that a stronger binding of the ligand to protein results in a higher score. Therefore, you enter a very high (or the string INFINITE) value for the target score, if you want to achieve the strongest binding possible. If the ligand is an inhibitor for example and you are looking for proteins with weak binding to this molecule, enter low values (or the string NEGATIVE-INFINITE). Of course, you can also enter values in between those extremes. However, this requires exact knowledge about the numbers, which differ from case to case. Test-runs are therefore unavoidable for this strategy. The target score defines the fitness of a protein. The closer the score of a protein is to the target score, the higher its fitness. With the optional tag *terminate-by-score-range* you can also force EvoDock to terminate, if the score of a protein falls into the given range around the target score. Earlier termination is also achievable with the optional tag *terminate-by-time*, which takes a value in the form of hhh:mm. Both termination cases do not force an immediate stop of EvoDock after the condition is fulfilled. It will still finish its last operation, because other optimal proteins might occur during this step as well.

After setting up the score related settings, the protein is the next important specification. Set the path to the (already prepared) pdb file with *-prot-path*. Next, we have to set which residues are targets of mutagenesis and which amino acids are allowed as substitutions for which position in the protein. For each residue of choice, add the tag *-res-id* followed by the chain (like A or B) and the id of the residue. In addition you have to write for each residue the tag *-substitutions*, which specify which amino acids are allowed for this position. To map the substitutions correctly to the residues, check if the appearing order of the corresponding tags is the same (so first substitution tag maps to first appearing res-id tag, second one to the second and so on). The *-substitution* comes with an optional parameter. Either, you left it empty so all amino acids are allowed, or you write the single-letter amini acid codes for all allowed amino acids, separated only (no spaces!) by a comma.

```
-init-pop-run-mode create-and-evolve 50
-task LIGAND-BINDING
-targetscore INFINITE
-prot-path input/2PQL.relaxed.pdb
-res-id A 3
-res-id A 111
-res-id A 135
-substitutions
-substitutions D,L
-substitutions

-mutate MutateByPyRosetta
-apply DockByPyRosetta
-score ScoreByPyRosetta
```

**Figure 3:** ***Example of an initial settings file.***
*These are the required specifications entered in an initial settings file. The shown settings create an initial population of size 50 and start an evolution on the task of ligand-binding afterwards. The residues with the ids 3, 111, and 135 from chain 'A' of the input protein are chosen for substitutions to create mutant variants, where residue 111 is substituted either with aspartic acid or with leucine. The other residues can receive any substitution. Higher fitness-scores are preferred. At the end, the used pipeline modules are specified.*

Finally, you have to enter the paths to the pipeline modules with *-mutate*, *-apply*, and *-score*, as well as specify the run-mode of EvoDock. Use *-init-pop-run-mode* to start and run EvoDock with a completely new population and proceed with the full algorithm

(create-and-evolve), create the new population only and stop (create-and-quit), or to load a previously generated initial population and perform the algorithm on it (load-and-evolve).

**Table 3:** *Optional specifications of the initial settings file.*

| Specification | Description |
|---|---|
| -terminate-by-score-range $n$ | Specify a range around the target score for which the evolution terminates, if any score is within it. |
| -terminate-by-time *time* | Specify a time in the format 'hhh:mm', where h and m stand for hours and minutes respectively. If the runtime exceeds the set time, the evolution terminates. |
| -include-mutant *mutant* | Adds the specified mutant to the initial population when generating it at random. The *mutant* parameter is a list of single-letter-coded amino acids. The single letters are separated only by a comma and the number of AAs must match the number of specified residues with '-res-id'. |
| -symmetry *a b* | *a*: Chain that is given in res-id, *b*: Chain id that replaces *a*. |
| -look-up-table-path *file-path* | Specify path to a text file with list of individuals, like a history. |
| –use-specific-mutate-out *TRUE/FALSE* | Specify, if the module specific output of the mutagenesis modules is used directly as input for the application, if both are compatible. If FALSE or if both modules are not compatible for this feature, PDB files are generated and are used as input. This would require more disk space and computation time. By default, this is TRUE. |
| -use-existent-mutate-out-paths *folder-path* | Specify path to a directory with already existing mutagenesis output, which are used as input for the application module, if available. The files must be named according to the mutations. An Example: 'Mutant_D_G1.pdb', where 'D' is the amino acid at first specified position and 'G' at second. Numbers must be given and are used as suffix for differentation of alternative structures of this mutant. If a file for a certain mutant is not existent, the strucutre will be generated via the mutagenesis module. |
| -skip-application *TRUE/FALSE* | If TRUE, the application module will be skipped. The evaluation module must be able to handle this case. By default, this is FALSE. |
| -cpu $n$ | Specify the number of cores used for parallel computation. If $n$ is 'MAX', all available cores are used. |
| -seed *arg* | Set the seed of the random generator for the run. Affects only random choices of the evolution process, does not affect pipeline modules. |
| -brute-force *TRUE/FALSE* | Set if new random individuals are generated via brute force if the recursion limit was reached once. By default, this is FALSE. |

For further control you can specify also the number of used *-cpu*, a *-seed* for the

random number generator (affects only randomness of the EvolutionModule, not of PyRosetta or other pipeline modules), existent pdb files for certain mutants, skip the application module, specify symmetry, add certain mutants, or specify a *-look-up-table-path* to give a list of pre-calculated scores for certain mutants. For all of these options, check Table 3.

Module specific options are listed in Table TODO and are explained in section 4.4.

## 4.3 Writing routine file

The routine file is a text file, which contains the routine for the Evolutionary Algorithm. It basically describes the procedure the algorithm has to follow and does therefore set the strategy for the EA. EvoDock terminates either after an optional termination condition was set and fulfilled or if the routine is over. Use the commands listed in Table 4, where each command is written in its own line. Avoid additional spaces after or prior a command. You can separate commands by blank lines and use also commands, which begin with a #. Keep in mind, however, that blank lines are also indicators for the loop command to end the range of the loop command.

**Table 4:** *Specifications of the routine file.*

| Specification | Description |
|---|---|
| mutate $m$ | Selects for each individual of the population a random gene and assigns a new random allele to it from the allowed allele-pool for this gene. This process is repeated $m$ times, but can lead to less than $m$ offsprings per individual, if no new gene sets are possible. |
| mutate+ $m$ | Works like 'mutate $m$', but additionally new child individuals move only into the population, if their fitness is higher than their parent's fitness. |
| recombine $m$ | Selects for each individual $m$ mating partners, to perform a uniform recombination with. |
| recombine-classic $m$ | Works like 'recombine $m$', but performs a classic recombination. |
| select $k$ | Selects a total of $k$ individuals from the population and keeps only these for the next generation. Performs elite selection. |
| select $p$ | Selects a fraction of $p$ individuals from the total population and keeps only these for the next generation. Performs elite selection. |
| select $k$ $r$ | Selects a total of $k$ individuals from the population and keeps only these for the next generation, where a total of $r$ of this selection is not chosen by best fitness but at random from the remaining population. |
| select $p$ $q$ | Selects a fraction of $p$ individuals from the total population and keeps only these for the next generation, where a fraction of $q$ of this selection is not chosen by best fitness but at random from the remaining population. |
| loop $n$ | Set a loop starting at this line and ending at the next blank line or at the end of the document. All lines in between are repeated $n$ times. Nested loops are not possible. |

Since it is not trivial to write the optimal routine, you can use the default routine file provided in the git repository. In short, mutate commands are essential to explore new

amino acids at a given position. Recombination can help to keep beneficial mutations if many residues are mutable, but has nearly no effect if only a few residues are modified. The selection differs depending on the datatype of the parameter. If you enter an integer (select k), you select k individuals. Entering a float number (select p), selects a fraction of the total population. In addition, it is possible to not only select the best individuals (so-called elite selection) but to select also some non-fittest individuals at random. This is often helpful, since the elite selection might lead to local optima, from which escape by a single mutation is impossible.

## 4.4 Module specific usage

The modules, using PyRosetta, a configurable as well. If you are not familiar with Rosetta/PyRosetta, you can use also the recommended and default settings.

**Table 5:** *Module specific commands in the initial settings file for the mutagenesis model "MutateByPyRosetta".*

| Command | Description |
|---|---|
| -model-param-mutate -score-function *arg* | Specify the used score function, where *arg* is either the name of a Rosetta score function of the path to a text file containing information about the score function. |
| -model-param-mutate -relax *mode* | Specify if the input protein will be relaxed at the start of the application, where *mode* is either 'FAST-RELAX', 'CLASSIC-RELAX, 'BOTH' or 'NONE'. 'BOTH' creates two separate, independent poses and is used for benchmark. |
| -model-param-mutate -extra-relax *mode* | Specify if each pose is relaxed after the mutagenesis and applying movers, where *mode* is either 'FAST-RELAX', 'CLASSIC-RELAX or 'NONE'. |
| -model-param-mutate -make-poses $n$ | Specify number of poses generated per mutant. |
| -model-param-mutate -save-pdb $b$ | Specify if each resulting pose is saved as a PDB file. |
| -model-param-mutate -rotamer-mover *mover* | Set the rotamer mover to one of the available movers. |
| -model-param-mutate -backbone-mover *mover* | Set the backbone mover to one of the available movers. |
| -model-param-mutate -pack-radius $r$ | Specify the radius $r$ in $\mathring{A}$ around each modified residue, within all residues are repacked. |
| -model-param-mutate -rotamer-moves-number $n$ | Specify the number of moves applied by the rotamer mover. |
| -model-param-mutate -backbone-moves-number $n$ | Specify the number of moves applied by the backbone mover. |
| -model-param-mutate -set-kT $kT$ | Set $kT$ value for backbone mover. |
| -model-param-mutate -set-n-moves $n$ | Set *n-moves* value for backbone mover. |

**Table 6:** *Module specific commands in the initial settings file for the application model "DockByPyRosetta".*

| Command | Description |
|---|---|
| -model-param-apply -score-function *arg* | Specify the used score function, where *arg* is either the name of a Rosetta score function of the path to a text file containing information about the score function. |
| -model-param-apply -xml-protocol-path *file-path* | Specify the path to the used Rosettta XML protocol. |
| -model-param-apply -xml-substitution *subst value* | Specify with which value each occurrence of *subst* in the XML protocol must be substituted. |
| -model-param-apply -model-param-mutate -make-poses $n$ | Specify number of poses generated per input pose. |
| -model-param-apply -model-param-mutate -save-pdb $b$ | Specify if each resulting pose is saved as a PDB file. |

**Table 7:** *Module specific commands in the initial settings file for the evaluation model "ScoreByPyRosetta".*

| Command | Description |
|---|---|
| -model-param-mutate -weight-mutate $w$ | Specifiy the weight of scores from mutagenesis results. |
| -model-param-mutate -weight-apply $w$ | Specifiy the weight of scores from application results. |

# 5 How to expand EvoDock?

How to modify modules, nested application modules, possible additional tasks: thermostability, crystallizability,

# 6 Implementation

```
Entries in init_pop_via_mutate: 240
['C', 'A', 'V'];952.7160022385468
['L', 'G', 'G'];951.0251070393583
['N', 'Q', 'C'];950.5202081039183
['N', 'C', 'G'];950.2411500186112
['R', 'G', 'D'];950.2166224763503
['A', 'S', 'T'];949.7784104598692
['S', 'H', 'A'];949.6960389914068
...
['G', 'P', 'K'];876.7027176904596
```

**Figure 4:** ***Example of an population file.***
*Shown is an extract from an output text file containing data of an initial population. The first line contains additional text information, while each other line contains the list of AAs and the fitness value. The population list is sorted, so the top seven and the worst individual are depicted.*

# 7 Contact

EvoDock was developed by Maximilian Edich at Bielefeld University in 2020. For questions regarding the usage and further development of EvoDock, please contact him:
medich@PHYSnet.uni-hamburg.de