AutoDock Vina Documentation

Release 1.2.0

['Center of Computational Structural Biology (CCSB) - Scripps Re

Sep 12, 2023

MANUAL

AutoDock Vina is one of the **fastest** and **most widely used open-source** docking engines. It is a turnkey computational docking program that is based on a simple scoring function and rapid gradient-optimization conformational search. It was originally designed and implemented by Dr. Oleg Trott in the Molecular Graphics Lab, and it is now being maintained and develop by the Forli Lab at The Scripps Research Institute.

Vina's design philosophy is not to require the user to understand its implementation details, tweak obscure search parameters, cluster results or know advanced algebra (quaternions). All that is required is the structures of the molecules being docked and the specification of the search space including the binding site. Calculating grid maps and assigning atom charges is not needed (when using Vina or Vinardo forcefields). Like in AutoDock 4, some receptor side chains can be chosen to be treated as flexible during docking. Additionally, it takes advantage of multiple CPUs or CPU cores on your system to significantly shorten its running time, which makes it faster than AutoDock 4 by *orders of magnitude*. Moreover, it significantly improves the average accuracy of the binding mode predictions compared to AutoDock 4.

CHAPTER

ONE

AVAILABILITY

AutoDock Vina can be easily installed with all its dependencies using *pip* or *conda* package managers.

All **source code** is available under the Apache License, version 2.0 from github.com/ccsb-scripps/AutoDock-Vina and the Python Package index pypi.org/project/Vina.

CHAPTER

TWO

PARTICIPATING

Please report bugs or enhancement requests through the Issue Tracker.

AutoDock Vina is **open source** and welcomes your contributions. Fork the repository on GitHub and submit a pull request. Participate on the developer mailing list.

2.1 About AutoDock Vina

AutoDock Vina (Vina) is one of the docking engines in the AutoDock Suite, together with **AutoDock4** (AD4), **AutoDockGPU**, **AutoDockFR**, and **AutoDock-CrankPep**, and arguably among the **most widely used and successful docking engines**. The reasons for this success are mostly due to its ease of use and its speed (up to 100x faster than AD4), when compared to the other docking engines in the suite and elsewhere, as well as being open source. The source code is available at https://github.com/ccsb-scripps/AutoDock-Vina.

Research groups around the world have modified and built upon the Vina source code, improving the search algorithm (**QuickVina2**), made the interface more user friendly and allow modification of scoring terms through the user interface (**Smina**), and improved the scoring function for carbohydrate docking (**Vina-Carb**), halogen bonds (**VinaXB**) and created an improved scoring function (**Vinardo**).

Over the years many specialized docking methods were developed for AutoDock 4 but were never ported to AutoDock Vina. This happened because AutoDock 4 allows the user to modify a large number of docking parameters, providing direct access to some of the engine internals, with benefits in terms of flexibility and adaptability. Conversely, the Vina code is highly optimized and specialized, and one of its hallmarks is the very limited amount of user input necessary to perform a docking, making it harder to implement additional functionality, which requires changes to the source code. However, some of the specialized docking methods are now available in Vina (v1.2.x) such as **macrocyclic flexibility**, **specialized metal coordination models**, **explicit waters hydrated** and **simultaneous docking of multiple ligands**. Vina was further extended by adding **Python bindings** to facilitate programmatic access to the docking engine functions.

2.2 Installation

Warning: Currently, the python bindings and the binary installation are two separate processes. The installation of the python bindings does not include the Vina executable, and vice versa.

2.2.1 Unix- and Linux-based OS

Vina is expected to work on x86 and compatible 64-bit Linux systems. The executable for the latest release are available here: https://github.com/ccsb-scripps/AutoDock-Vina/releases.

Running:

```
./vina_1.2.0_linux_x86_64 --help
```

If the executable is in your PATH, you can just type "vina –help" instead.

2.2.2 macOS

Vina is expected to work on macOS 10.15 (Catalina) and newer. The executable for the latest release are available here: https://github.com/ccsb-scripps/AutoDock-Vina/releases.

Running:

```
./vina_1.2.0_macos_x86_64 --help
```

If the executable is in your PATH, you can just type "vina –help" instead.

2.2.3 Python bindings

Currently working on Linux and Mac. We did not test on Windows.

AutoDock Vina installation using pip:

Note: When using `pip`, it's good pratice to use a virtual environment and also the easiest solution (see `meeko` in *Software requirements*). An example with the Conda package manager is available further down.

```
$ pip install -U numpy vina
```

AutoDock Vina installation in a Conda environment:

The Anaconda Python distribution, which can be download from https://docs.continuum.io/anaconda/install. This is a Python distribution specially designed for scientific applications, with many of the most popular scientific packages preinstalled. Alternatively, you can use Miniconda, which includes only Python itself, plus the Conda package manager.

1. Begin by installing the most recent 64 bit, Python 3.x version of either Anaconda or Miniconda
2. Create a dedicated environment for `AutoDock Vina`. This environment can be re-used for installing `meeko` (see *Software requirements*):

```
$ conda create -n vina python=3
$ conda activate vina
$ conda config --env --add channels conda-forge
```

3. And type the following command to install `NumPy` and `AutoDock Vina`:

```
$ conda install -c conda-forge numpy swig boost-cpp sphinx sphinx_rtd_theme
$ pip install vina
```

2.2.4 Building from Source

Warning: Building Vina from source is NOT meant to be done by regular users!

- **Step 1: Install a C++ compiler suite**
  - Ubuntu/Debian: `sudo apt-get install build-essential`
  - macOS: Install Xcode from the AppStore and the Command Line Tools (CLT) from the terminal `xcode-select --install`
- **Step 2: Install Boost and SWIG**
  - Ubuntu/Debian: `sudo apt-get install libboost-all-dev swig`
  - macOS (with Homebrew): `brew install boost swig`
- Step 3: **Build Vina**

  Start by downloading the lastest version of `AutoDock Vina` from github:

```
$ git clone https://github.com/ccsb-scripps/AutoDock-Vina
```

  To compile the binary (you might need to customize the Makefile by setting the paths to the Boost library):

```
$ cd AutoDock-Vina/build/linux/release
$ make
```

  To compile the Python bindings:

  Note: The `Conda` package manager is used here to easily install the several dependencies needed to build the `Autodock-Vina` python bindings (see above how to create a dedicated environment).

```
$ conda activate vina
$ cd AutoDock-Vina/build/python
$ conda install -c conda-forge numpy boost-cpp swig
$ rm -rf build dist *.egg-info (to clean previous installation)
$ python setup.py build install
```

2.3 Frequently Asked Questions

- **How accurate is AutoDock Vina?**

The predictive accuracy varies a lot depending on the target, so it makes sense to evaluate AutoDock Vina against your particular target first, if you have known actives, or a bound native ligand structure, before ordering compounds. While evaluating any docking engine in a retrospective virtual screen, it might make sense to select decoys of similar size, and perhaps other physical characteristics, to your known actives.

- **What is the difference between AutoDock Vina and AutoDock 4?**

AutoDock 4 (and previous versions) and AutoDock Vina were both developed in the Molecular Graphics Lab at The Scripps Research Institute. AutoDock Vina inherits some of the ideas and approaches of AutoDock 4, such as treating docking as a stochastic global opimization of the scoring function, precalculating grid maps (Vina does that internally),

and some other implementation tricks, such as precalculating the interaction between every atom type pair at every distance. It also uses the same type of structure format (PDBQT) for maximum compatibility with auxiliary software.

However, the source code, the scoring funcion and the actual algorithms used are brand new, so it's more correct to think of AutoDock Vina as a new "generation" rather than "version" of AutoDock. The performance was compared in the original publication [*], and on average, AutoDock Vina did considerably better, both in speed and accuracy. However, for any given target, either program may provide a better result, even though AutoDock Vina is more likely to do so. This is due to the fact that the scoring functions are different, and both are inexact.

- **What is the difference between AutoDock Vina and AutoDock Tools?**

AutoDock Tools is a module within the MGL Tools software package specifically for generating input (PDBQT files) for AutoDock or Vina. It can also be used for viewing the results.

- **Can I dock two proteins with AutoDock Vina?**

You might be able to do that, but AutoDock Vina is designed only for receptor-ligand docking. There are better programs for protein-protein docking.

- **Will Vina run on my 64-bit machine?**

Yes. By design, modern 64-bit machines can run 32-bit binaries natively.

- **Why do I get "can not open conf.txt" error? The file exists!**

Oftentimes, file browsers hide the file extension, so while you think you have a file "conf.txt", it's actually called "conf.txt.txt". This setting can be changed in the control panel or system preferences.

You should also make sure that the file path you are providing is correct with respect to the directory (folder) you are in, e.g. if you are referring simply to conf.txt in the command line, make sure you are in the same directory (folder) as this file. You can use ls or dir commands on Linux/MacOS and Windows, respectively, to list the contents of your directory.

- **Why do I get "usage errors" when I try to follow the video tutorial?**

The command line options changed somewhat since the tutorial has been recorded. In particular, "–out" replaced "–all".

- **Vina runs well on my machine, but when I run it on my exotic Linux cluster, I get a "boost thread resource" error. Why?**

Your Linux cluster is [inadvertantly] configured in such a way as to disallow spawning threads. Therefore, Vina can not run. Contact your system administrator.

- **Why is my docked conformation different from what you get in the video tutorial?**

The docking algorithm is non-deterministic. Even though with this receptor-ligand pair, the minimum of the scoring function corresponds to the correct conformation, the docking algorithm sometimes fails to find it. Try several times and see for yourself. Note that the probability of failing to find the mininum may be different with a different system.

- **My docked conformation is the same, but my energies are different from what you get in the video tutorial. Why?**

The scoring function has changed since the tutorial was recorded, but only in the part that is independent of the conformation: the ligand-specific penalty for flexibility has changed.

- **Why do my results look weird in PyMOL?**

PDBQT is not a standard molecular structure format. The version of PyMOL used in the tutorial (0.99rc6) happens to display it well (because PDBQT is somewhat similar to PDB). This might not be the case for newer versions of PyMOL.

- **Any other way to view the results?**

You can also view PDBQT files in PMV (part of MGL Tools), or convert them into a different file format (e.g. using AutoDock Tools, or with "save as" in PMV)

- **How big should the search space be?**

As small as possible, but not smaller. The smaller the search space, the easier it is for the docking algorithm to explore it. On the other hand, it will not explore ligand and flexible side chain atom positions outside the search space. You should probably avoid search spaces bigger than 30 x 30 x 30 Angstrom, unless you also increase "–exhaustiveness".

- **Why am I seeing a warning about the search space volume being over 27000 Angstrom^3?**

This is probably because you intended to specify the search space sizes in "grid points" (0.375 Angstrom), as in AutoDock 4. The AutoDock Vina search space sizes are given in Angstroms instead. If you really intended to use an unusually large search space, you can ignore this warning, but note that the search algorithm's job may be harder. You may need to increase the value of the exhaustiveness to make up for it. This will lead to longer run time.

- **The bound conformation looks reasonable, except for the hydrogens. Why?**

AutoDock Vina actually uses a united-atom scoring function, i.e. one that involves only the heavy atoms. Therefore, the positions of the hydrogens in the output are arbitrary. The hydrogens in the input file are used to decide which atoms can be hydrogen bond donors or acceptors though, so the correct protonation of the input structures is still important.

- **What does "exhaustiveness" really control, under the hood?**

In the current implementation, the docking calculation consists of a number of independent runs, starting from random conformations. Each of these runs consists of a number of sequential steps. Each step involves a random perturbation of the conformation followed by a local optimization (using the Broyden-Fletcher-Goldfarb-Shanno algorithm) and a selection in which the step is either accepted or not. Each local optimization involves many evaluations of the scoring function as well as its derivatives in the position-orientation-torsions coordinates. The number of evaluations in a local optimization is guided by convergence and other criteria. The number of steps in a run is determined heuristically, depending on the size and flexibility of the ligand and the flexible side chains. However, the number of runs is set by the exhaustiveness parameter. Since the individual runs are executed in parallel, where appropriate, exhaustiveness also limits the parallelism. Unlike in AutoDock 4, in AutoDock Vina, each run can produce several results: promising intermediate results are remembered. These are merged, refined, clustered and sorted automatically to produce the final result.

- **Why do I not get the correct bound conformation?**

It can be any of a number of things:

1. If you are coming from AutoDock 4, a very common mistake is to specify the search space in "points" (0.375 Angstrom), instead of Angstroms.
2. Your ligand or receptor might not have been correctly protonated.
3. Bad luck (the search algorithm could have found the correct conformation with good probability, but was simply unlucky). Try again with a different seed.
4. The minimum of the scoring function correponds to the correct conformation, but the search algorithm has trouble finding it. In this case, higher exhaustiveness or smaller search space should help.
5. The minimum of the scoring function simply is not where the correct conformation is. Trying over and over again will not help, but may occasionally give the right answer if two wrongs (inexact search and scoring) make a right. Docking is an approximate approach.
6. Related to the above, the culprit may also be the quality of the X-ray or NMR receptor structure.
7. If you are not doing redocking, i.e. using the correct induced fit shape of the receptor, perhaps the induced fit effects are large enough to affect the outcome of the docking experiment.
8. The rings can only be rigid during docking. Perhaps they have the wrong conformation, affecting the outcome.
9. You are using a 2D (flat) ligand as input.

10. The actual bound conformation of the ligand may occasionally be different from what the X-ray or NMR structure shows.
11. Other problems

- **How can I tweak the scoring function?**

You can change the weights easily, by specifying them in the configuration file, or in the command line. For example

```
vina --weight_hydrogen -1.2 ...
```

doubles the strenth of all hydrogen bonds.

- **Functionality that would allow the users to create new atom and pseudo-atom types, and specify their own interaction functions is planned for the future.**

This should make it easier to adapt the scoring function to specific targets, model covalent docking and macro-cycle flexibility, experiment with new scoring functions, and, using pseudo-atoms, create directional interaction models.

Stay tuned to the AutoDock mailing list, if you wish to be notified of any beta-test releases.

- **Why don't I get as many binding modes as I specify with "–num_modes"?**

This option specifies the maximum number of binding modes to output. The docking algorithm may find fewer "interesting" binding modes internally. The number of binding modes in the output is also limited by the "energy_range", which you may want to increase.

- **Why don't the results change when I change the partial charges?**

AutoDock Vina ignores the user-supplied partial charges. It has its own way of dealing with the electrostatic interactions through the hydrophobic and the hydrogen bonding terms. See the original publication [*] for details of the scoring function.

- **I changed something, and now the docking results are different. Why?**

Firstly, had you not changed anything, some results could have been different anyway, due to the non-deterministic nature of the search algorithm. Exact reproducibility can be assured by supplying the same random seed to both calculations, but only if all other inputs and parameters are the same as well. Even minor changes to the input can have an effect similar to a new random seed. What does make sense discussing are the statistical properties of the calculations: e.g. "with the new protonation state, Vina is much less likely to find the correct docked conformation".

- **How do I use flexible side chains?**

You split the receptor into two parts: rigid and flexible, with the latter represented somewhat similarly to how the ligand is represented. See the section "Flexible Receptor PDBQT Files" of the AutoDock4.2 User Guide (page 14) for how to do this in AutoDock Tools. Then, you can issue this command: vina –config conf –receptor rigid.pdbqt –flex side_chains.pdbqt –ligand ligand.pdbqt. Also see this write-up on this subject.

- **How do I do virtual screening?**

Please see the relevant section of the manual. Please note that a variety of docking management applications exist to assist you in this task.

- **I have ideas for new features and other suggestions.**

For proposed new features, we like there to be a wide consensus, resulting from a public discussion, regarding their necessity. Please consider starting or joining a discussion on the AutoDock mailing list.

- **Will you answer my questions about Vina if I email or call you?**

No. Vina is community-supported. There is no obligation on the authors to help others with their projects. Please see this page for how to get help.

2.4 Citations

2.4.1 AutoDock Vina

AutoDock Vina is a scientific software that is described in academics publications. Please cite these papers when you used AutoDock Vina in your work:

- Eberhardt, J., Santos-Martins, D., Tillack, A.F., Forli, S. (2021). AutoDock Vina 1.2.0: New Docking Methods, Expanded Force Field, and Python Bindings. Journal of Chemical Information and Modeling.
- Trott, O., & Olson, A. J. (2010). AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. Journal of computational chemistry, 31(2), 455-461.

2.4.2 Related publications

- Santos-Martins, D., Eberhardt, J., Bianco, G., Solis-Vasquez, L., Ambrosio, F. A., Koch, A., & Forli, S. (2019). D3R Grand Challenge 4: prospective pose prediction of BACE1 ligands with AutoDock-GPU. Journal of Computer-Aided Molecular Design, 33(12), 1071-1081.
- Forli, S., Huey, R., Pique, M. E., Sanner, M. F., Goodsell, D. S., & Olson, A. J. (2016). Computational protein–ligand docking and virtual drug screening with the AutoDock suite. Nature protocols, 11(5), 905-919.
- Santos-Martins, D., Forli, S., Ramos, M. J., & Olson, A. J. (2014). AutoDock4Zn: an improved AutoDock force field for small-molecule docking to zinc metalloproteins. Journal of chemical information and modeling, 54(8), 2371-2379.
- Forli, S., & Olson, A. J. (2012). A force field with discrete displaceable waters and desolvation entropy for hydrated ligand docking. Journal of medicinal chemistry, 55(2), 623-638.
- Forli, S., & Botta, M. (2007). Lennard-Jones potential and dummy atom settings to overcome the AUTODOCK limitation in treating flexible ring systems. Journal of chemical information and modeling, 47(4), 1481-1492.

2.5 Changes

2.5.1 1.2.0

- Refactoring of the Vina code to be able to be used as a library
- Can load external maps
- Support of simultaneous docking of multiple ligands and batch mode for virtual screening
- Support of macrocycle molecules
- Addition of the hydrated docking protocol
- Added AutoDock4 and Vinardo forcefields
- Added Python bindings for Python 3

2.5.2 1.1.2

- Bug fix: the affinities reported with flexible side chains were sometimes incorrect. (This problem had no effect on the predicted binding modes or their relative ranking, or on rigid receptor docking)
- Bug fix: the individual term values, before weighting, reported only with "score_only", were often wrong. (This problem had no effect on the predicted binding modes or affinities)
- The predicted binding modes are made less redundant by requiring that no two reported modes differ by less than 1 Angstrom RMSD(U.B.), counting all movable heavy atoms, including those in the side chains. Previously, Vina avoided this kind of redundancy during the actual docking, but made no such guarantee w.r.t. the output because of the subsequent refinement stage that could move different binding modes closer
- Bug fix: in some very unusual cases, numerical rounding errors would accumulate, leading, internally, to a distorted ligand structure. The self-checks in Vina would catch this and make the program quit rather than write invalid output. This numerical error accumulation is believed to have been fixed in this update
- A warning is added when the search space is greater than 27000 Angstrom^3 in volume, as confusing Angstroms with "grid points" is a frequent user error

2.5.3 1.1.1

- Source code released
- License changed to Apache
- Extensive code clean-up that however should not have any effect on the end users
- Unix binary packages changed to simple archives

2.5.4 1.1.0

- Added "advanced options" intended to be primarily used by people interested in methods development.

  These should allow scoring without minimization; performing local optimization only; randomizing the input with no search (this is useful for testing docking software); changing the weights from their default values; displaying the individual contributions to the intermolecular score, before weighting (these are shown with "–score_only")

2.5.5 1.0.3

- Mac OS X 10.6 (Snow Leopard) support added
- A bug related to (very rarely) not finding any modes when the "num_modes" setting is low (e.g. 1) has been fixed.

2.5.6 1.0.2

- Fixed a bug that excluded conformations in which only a hydrogen atom was outside the search space.

2.5.7 1.0.1

- When the search space is too small or misplaced, so that the ligand and flexible side chain conformations can not be found within it, Vina will no longer produce any "binding modes" and print a warning instead
- Minor improvements in user-friendliness of the error messages.

2.5.8 1.0

- Support for earlier versions of Linux added, such as Debian 4.0
- Friendlier and more specific error messages for PDBQT syntax errors
- "all" parameter is now called "out" and is always enabled. Individual binding modes are no longer written separately. If needed, the multimodel output can be split into individual models using a separate program called "vina_split", included in the distribution
- The default "num_modes" value changed to 9.

2.6 Software requirements

For those tutorials, the ADFR software suite, providing a number of software tools for automated docking and peripheral tasks, and the Python package Raccoon-lite, for preparing macrocycle for example, are necessary.

2.6.1 ADFR software suite

Warning: macOS users please note that ADFR software suite is NOT working under Catalina (10.15). We will send email to the mailing list if and when ADFR software suite or an alternative will be available for this version of macOS. If you already are using Catalina, we recommend install VirtualBox and running ADFR software suite inside the virtual box.

The ADFR software suite was developed in the Sanner lab at the Center for Computational Structural Biology (CCSB) formerly known as the Molecular Graphics Laboratory (MGL) of The Scripps Research Institute for visualization and analysis of molecular structures. You can find more information about the ADFR software suite installation process here: https://ccsb.scripps.edu/adfr/downloads. The current version contains the following tools for docking:

- ADFR v1.2 and associate scripts
- AGFR v1.2
- AutoSite v1.0 and v1.1
- ADCP v1.0
- AutoGrid4.2
- prepare_ligand
- prepare_receptor

Moreover, the ADFR software suite provides a number of software tools for automated docking and peripheral tasks. These tools are implemented using the Python, C++ and C programming languages and a re-usable component philosophy. To avoid Python packages mismatches we opted to shift ADFR suite with a self-contain Python interpreter that is isolated from the default Python interpreter installed on your computer (except for Windows installations). Details about the implementation and packages provided by the ADFR software suite can be found here: https://ccsb.scripps.edu/adfr/implementation

Citations:

- Zhang, Y., Forli, S., Omelchenko, A., & Sanner, M. F. (2019). AutoGridFR: Improvements on AutoDock Affinity Maps and Associated Software Tools. Journal of Computational Chemistry, 40(32), 2882-2886.
- Zhang, Y., & Sanner, M. F. (2019). AutoDock CrankPep: combining folding and docking to predict protein–peptide complexes. Bioinformatics, 35(24), 5121-5127.
- Ravindranath, P. A., & Sanner, M. F. (2016). AutoSite: an automated approach for pseudo-ligands prediction—from ligand-binding sites identification to predicting key ligand atoms. Bioinformatics, 32(20), 3142-3149.
- Ravindranath, P. A., Forli, S., Goodsell, D. S., Olson, A. J., & Sanner, M. F. (2015). AutoDockFR: advances in protein-ligand docking with explicitly specified binding site flexibility. PLoS computational biology, 11(12), e1004586.
- Zhao, Y., Stoffler, D., & Sanner, M. (2006). Hierarchical and multi-resolution representation of protein flexibility. Bioinformatics, 22(22), 2768-2774.

2.6.2 Meeko

The Python package `meeko` is a new type of package developped in the Forli lab also at the Center for Computational Structural Biology (CCSB). It provides tools covering other docking aspects not handled by the ADFR software suite. This package provides addtionnal tools for the following docking protocols:

- Hydrated docking
- Macrocycles

Meeko installation using pip:

Note: When using `pip`, it's good pratice to use a virtual environment and also the easiest solution. An example with the Conda package manager is available further down.

Warning: OpenBabel executable and library must be installed first, see installation instruction.

```
$ pip install -U numpy openbabel meeko
```

If the installation was successful, you should now be able to access to the following command from your terminal by typing:

- mk_prepare_ligand.py

Meeko installation in a Conda environment:

Note: See instructions in *Installation* on how to setup and create an dedicated `Conda` environment.

Type the following command to install `NumPy`, `OpenBabel` and `meeko`:

```
$ conda activate vina
$ conda install python=3.9.7    # not strictly needed but prevents RDKit incompatibility
$ conda install -c conda-forge numpy openbabel scipy rdkit
$ pip install meeko
```

2.7 Basic docking

Let's start with our first example of docking, where the typical usage pattern would be to dock a single molecule into a rigid receptor. In this example we will dock the approved anticancer drug imatinib (Gleevec; PDB entry 1iep) in the structure of c-Abl using AutoDock Vina. The target for this protocol is the kinase domain of the proto-oncogene tyrosine protein kinase c-Abl. The protein is an important target for cancer chemotherapy—in particular, the treatment of chronic myelogenous leukemia.

Note: This tutorial requires a certain degree of familiarity with the command-line interface. Also, we assume that you installed the ADFR software suite as well as the meeko Python package.

Note: The materials present is this tutorial can be also found here: https://www.nature.com/articles/nprot.2016.051. If you are using this tutorial for your works, you can cite the following paper:

- Forli, S., Huey, R., Pique, M. E., Sanner, M. F., Goodsell, D. S., & Olson, A. J. (2016). Computational protein–ligand docking and virtual drug screening with the AutoDock suite. Nature protocols, 11(5), 905-919.

2.7.1 Materials for this tutorial

For this tutorial, all the basic material are provided and can be found in the `AutoDock-Vina/example/basic_docking/data` directory (or on GitHub). If you ever feel lost, you can always take a look at the solution here: `AutoDock-Vina/example/basic_docking/solution`. All the Python scripts used here (except for `prepare_receptor` and `mk_prepare_ligand.py`) are located in the `AutoDock-Vina/example/autodock_scripts` directory, alternatively you can also find them here on GitHub.

2.7.2 1. Preparing the receptor

During this step, we will create a PDBQT file of our receptor containing only the polar hydrogen atoms as well as partial charges. For this step, we will use the `prepare_receptor` command tool from the ADFR Suite. As a prerequisite, a receptor coordinate file must contain all hydrogen atoms. If hydrogen atoms are absent in the protein structure file, you can add the `-A "hydrogens"` flag. Many tools exist to add missing hydrogen atoms to a protein, one popular choice would be to use REDUCE. If you are using experimental structures (for instance, from the Protein Data Bank), use a text editor to remove waters, ligands, cofactors, ions deemed unnecessary for the docking. This file contains the receptor coordinates taken from PDB entry 1iep.

```
$ prepare_receptor -r 1iep_receptorH.pdb -o 1iep_receptor.pdbqt
```

Other options are available for `prepare_receptor` by typing `prepare_receptor -h`. If you are not sure about this step, the output PDBQT file `1iep_receptor.pdbqt` is available in `solution` directory.

2.7.3 2. Preparing the ligand

This step is very similar to the previous step. We will also create a PDBQT file from a ligand molecule file (in MOL/MOL2 or SDF format) using the `Meeko` python package (see installation instruction here: *Software requirements*). For convenience, the file `1iep_ligand.sdf` is provided (see `data` directory). But you can obtain it directly from the PDB here: 1iep (see `Download instance Coordinates` link for the STI molecule). Since the ligand file does not include the hydrogen atoms, we are going to automatically add them.

Warning: We strongly advice you against using PDB format for preparing small molecules, since it does not contain information about bond connections. Please don't forget to always check the protonation state of your molecules before docking. Your success can sometimes hang by just an hydrogen atom. ;-)

```
$ mk_prepare_ligand.py -i 1iep_ligand.sdf -o 1iep_ligand.pdbqt
```

Other options are available for `mk_prepare_ligand.py` by typing `mk_prepare_ligand.py --help`. If you are not sure about this step, the output PDBQT file `1iep_ligand.pdbqt` is available in `solution` directory.

2.7.4 3. (Optional) Generating affinity maps for AutoDock FF

Now, we have to define the grid space for the docking, typically, a 3D box around a the potential binding site of a receptor. During this step, we will create the input file for AutoGrid4, which will create an affinity map file for each atom types. The grid parameter file specifies an AutoGrid4 calculation, including the size and location of the grid, the atom types that will be used, the coordinate file for the rigid receptor, and other parameters for calculation of the grids.

To prepare the gpf file for AutoGrid4, you can use the `prepare_gpf.py` command line tool.

```
$ pythonsh <script_directory>/prepare_gpf.py -l 1iep_ligand.pdbqt -r 1iep_receptor.pdbqt␣
↪-y
```

The option `-y` specifies that we want to center automatically the grid around the ligand. For more information about `prepare_gpf.py`, type `pythonsh prepare_gpf.py -h`. At the end you should obtain the following GPF file `1iep_receptor.gpf` containing those lines:

Listing 1: Content of the grid parameter file (**1iep_receptor.gpf**) for the receptor c-Abl (**1iep_receptor.pdbqt**)

```
npts 54 54 54                        # num.grid points in xyz
gridfld 1iep_receptor.maps.fld       # grid_data_file
spacing 0.375                        # spacing(A)
receptor_types A C OA N SA HD        # receptor atom types
ligand_types A C NA OA N HD          # ligand atom types
receptor 1iep_receptor.pdbqt         # macromolecule
gridcenter 15.190 53.903 16.917      # xyz-coordinates or auto
smooth 0.5                           # store minimum energy w/in rad(A)
map 1iep_receptor.A.map              # atom-specific affinity map
map 1iep_receptor.C.map              # atom-specific affinity map
map 1iep_receptor.NA.map             # atom-specific affinity map
map 1iep_receptor.OA.map             # atom-specific affinity map
map 1iep_receptor.N.map              # atom-specific affinity map
map 1iep_receptor.HD.map             # atom-specific affinity map
elecmap 1iep_receptor.e.map          # electrostatic potential map
```

(continues on next page)

(continued from previous page)

```
dsolvmap 1iep_receptor.d.map         # desolvation potential map
dielectric -0.1465                   # <0, AD4 distance-dep.diel;>0, constant
```

After creating the GPF file, and now we can use the `autogrid4` command to generate the different map files that will be used for the molecular docking:

```
$ autogrid4 -p 1iep.gpf -l 1iep.glg
```

From this command you should have generated the following files:

```
1iep_receptor.maps.fld       # grid data file
1iep_receptor.*.map          # affinity maps for A, C, HD, H, NA, N, OA atom types
1iep_receptor.d.map          # desolvation map
1iep_receptor.e.map          # electrostatic map
```

2.7.5 4. Running AutoDock Vina

The imatinib ligand used in this protocol is challenging, and Vina will occasionally not find the correct pose with the default parameters. Vina provides a parameter called `exhaustiveness` to change the amount of computational effort used during a docking experiment. The default exhaustiveness value is `8`; increasing this to `32` will give a more consistent docking result. At this point of the tutorial, you have the choice to decide to run the molecular docking using either the `AutoDock` forcefield (requires affinity maps, see previous step) or using the `Vina` forcefield (no need for affinity maps).

4.a. Using AutoDock4 forcefield

When using the AutoDock4 forcefield, you only need to provide the affinity maps and the ligand, while specifying that the forcefield used will be AutoDock4 using the option `--scoring ad4`.

```
$ vina  --ligand 1iep_ligand.pdbqt --maps 1iep_receptor --scoring ad4 \
        --exhaustiveness 32 --out 1iep_ligand_ad4_out.pdbqt
```

Running AutoDock Vina will write a PDBQT file called `1iep_ligand_ad4_out.pdbqt` contaning all the poses found during the molecular docking and also present docking information to the terminal window.

4.b. Using Vina forcefield

Contrary to AutoDock4, you don't need to precalculate the affinity grid maps with `autogrid4` when using the Vina forcefield. AutoDock Vina computes those maps internally before the docking. However, you still need to specify the center and dimensions (in Angstrom) of the grid space, as well as the receptor. Here, instead of specifying each parameters for the grid box using the arguments `--center_x, --center_y, --center_z` and `--size_x, --size_y, --size_z`, we will store all those informations in a text file `1iep_receptor_vina_box.txt`.

Listing 2: Content of the config file (**1iep_receptor_vina_box.txt**) for AutoDock Vina

```
center_x = 15.190
center_y = 53.903
center_z = 16.917
size_x = 20.0
```

(continues on next page)

(continued from previous page)

```
size_y = 20.0
size_z = 20.0
```

```
$ vina --receptor 1iep_receptor.pdbqt --ligand 1iep_ligand.pdbqt \
       --config 1iep_receptor_vina_box.txt \
       --exhaustiveness=32 --out 1iep_ligand_vina_out.pdbqt
```

Tip: Alternatively, you can use the Vinardo forcefield by adding the `--scoring vinardo` option.

Running AutoDock Vina will write a PDBQT file called `1iep_ligand_vina_out.pdbqt`.

2.7.6 5. Results

With `exhaustiveness` set to `32`, Vina will most often give a single docked pose with this energy. With the lower default exhaustiveness, several poses flipped end to end, with less favorable energy, may be reported.

Warning: Please don't forget that energy scores giving by the AutoDock and Vina forcefield are not comparable between each other.

5.a. Using AutoDock forcefield

The predicted free energy of binding should be about `-14 kcal/mol` for poses that are similar to the crystallographic pose.

```
Scoring function : ad4
Ligand: 1iep_ligand.pdbqt
Exhaustiveness: 32
CPU: 0
Verbosity: 1

Reading AD4.2 maps ... done.
Performing docking (random seed: -556654859) ...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************

mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -14.62          0          0
   2       -13.13      1.051      1.529
   3       -12.26      1.442      2.158
   4       -11.91      3.646       11.5
   5       -11.89      3.859      11.99
   6       -11.47      1.978      13.56
   7       -11.33      1.727      2.585
   8       -10.85      3.619      5.759
   9       -10.23      7.057       12.7
```

5.b. Using Vina forcefield

Using the vina forcefield, you should obtain a similar output from Vina with the best score around `-13 kcal/mol`.

```
Scoring function : vina
Rigid receptor: 1iep_receptor.pdbqt
Ligand: 1iep_ligand.pdbqt
Center: X 15.19 Y 53.903 Z 16.917
Size: X 20 Y 20 Z 20
Grid space: 0.375
Exhaustiveness: 32
CPU: 0
Verbosity: 1

Computing Vina grid ... done.
Performing docking (random seed: -131415392) ...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************

mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -12.92          0          0
   2       -10.97      3.012      12.42
   3       -10.79      3.713      12.19
   4       -10.69      3.913      12.36
   5       -10.32      2.538      12.64
   6       -9.464      2.916      12.53
   7       -9.204       1.35      2.025
   8       -9.137      1.596      2.674
   9       -8.637      3.969      12.69
```

2.8 Flexible docking

The lack of receptor flexibility is arguably the greatest limitation in these types of docking methods. However, AutoDock Vina allows some limited flexibility of selected receptor side chains. In this tutorial, we will describe the cross-docking of the imatinib molecule to c-Abl in PDB entry 1fpu, treating Thr315 as flexible.

Note: This tutorial requires a certain degree of familiarity with the command-line interface. Also, we assume that you installed the ADFR software suite as well as the meeko Python package.

Note: The materials present is this tutorial can be also found here: https://www.nature.com/articles/nprot.2016.051. If you are using this tutorial for your works, you can cite the following paper:

- Forli, S., Huey, R., Pique, M. E., Sanner, M. F., Goodsell, D. S., & Olson, A. J. (2016). Computational protein–ligand docking and virtual drug screening with the AutoDock suite. Nature protocols, 11(5), 905-919.

2.8.1 Materials for this tutorial

For this tutorial, all the basic material are provided and can be found in the `AutoDock-Vina/example/flexible_docking/data` directory (or on GitHub). If you ever feel lost, you can always take a look at the solution here: `AutoDock-Vina/example/flexible_docking/solution`. All the Python scripts used here (except for `prepare_receptor` and `mk_prepare_ligand.py`) are located in the `AutoDock-Vina/example/autodock_scripts` directory, alternatively you can also find them here on GitHub.

2.8.2 1. Preparing the flexible receptor

During this step, we are going to split the receptor coordinates into two PDBQT files: one for the rigid portion and one for the flexible side chains. As with the *Basic docking* tutorial, the method requires a receptor coordinate file that includes all hydrogen atoms. The file `1fpu_receptorH.pdb` is provided (see `<autodock-vina_directory>/example/flexible_docking/data` directory). It contains the receptor coordinates taken from PDB entry `1fpu`.

```
$ prepare_receptor -r 1fpu_receptorH.pdb -o 1fpu_receptor.pdbqt
$ pythonsh <script_directory>/prepare_flexreceptor.py -r 1fpu_receptor.pdbqt -s THR315
```

Other options are available for `prepare_flexreceptor.py` with the `-h` option. This will create two different files, one containing only the rigid part of the protein, and the other one containing Thr315 as flexible residue:

```
1fpu_receptor_rigid.pdbqt           # rigid part
1fpu_receptor_flex.pdbqt            # flexible sidechain of Thr315
```

If you are not sure about this step, the output PDBQT files `1fpu_receptor_rigid.pdbqt` and `1fpu_receptor_flex.pdbqt` are available in `solution` directory.

2.8.3 2. Prepare ligand

For the molecular docking with flexible sidechains, we will use the ligand file `1iep_ligand.pdbqt` from the previous tutorial *Basic docking*.

2.8.4 3. (Optional) Generating affinity maps for AutoDock FF

As well as for the docking with a fully rigid receptor, we need to generate a GPF file to precalculate the affinity map for each atom types. However, instead of using the full receptor, affinity maps will be calculated only for the rigid part of the receptor (`1fpu_receptor_rigid.pdbqt`).

To prepare the GPF file for the rigid part of the receptor:

```
$ pythonsh <script_directory>/prepare_gpf.py -l 1iep_ligand.pdbqt -r 1fpu_receptor_rigid.
↪pdbqt -y
```

Luckily for us, the structure `1iep` and `1fpu` are almost perfectly superposed already, so we can also center the grid around `1iep_ligand.pdbqt`. Otherwise, the center of the grid can be specified using the option `-p gridcenter='X,X,X'`.

Listing 3: Content of the grid parameter file (**1fpu_receptor_rigid.gpf**) for the receptor c-Abl (**1fpu_receptor_rigid.pdbqt**)

```
npts 54 54 54                          # num.grid points in xyz
gridfld 1fpu_receptor_rigid.maps.fld # grid_data_file
spacing 0.375                          # spacing(A)
receptor_types A C NA OA N SA HD     # receptor atom types
ligand_types A C NA OA N HD          # ligand atom types
receptor 1fpu_receptor_rigid.pdbqt   # macromolecule
gridcenter 15.190 53.903 16.917      # xyz-coordinates or auto
smooth 0.5                             # store minimum energy w/in rad(A)
map 1fpu_receptor_rigid.A.map        # atom-specific affinity map
map 1fpu_receptor_rigid.C.map        # atom-specific affinity map
map 1fpu_receptor_rigid.NA.map       # atom-specific affinity map
map 1fpu_receptor_rigid.OA.map       # atom-specific affinity map
map 1fpu_receptor_rigid.N.map        # atom-specific affinity map
map 1fpu_receptor_rigid.HD.map       # atom-specific affinity map
elecmap 1fpu_receptor_rigid.e.map    # electrostatic potential map
dsolvmap 1fpu_receptor_rigid.d.map   # desolvation potential map
dielectric -0.1465                     # <0, AD4 distance-dep.diel;>0, constant
```

To execute `autogrid4` using `1fpu_receptor_rigid.gpf`, run the folllowing command line:

```
$ autogrid4 -p 1fpu_receptor_rigid.gpf -l 1fpu_receptor_rigid.glg
```

You should obtain as well the following files:

```
1fpu_receptor.maps.fld       # grid data file
1fpu_receptor.*.map          # affinity maps for A, C, HD, NA, N, OA atom types
1fpu_receptor.d.map          # desolvation map
1fpu_receptor.e.map          # electrostatic map
```

2.8.5 4. Running AutoDock Vina

4.a. Using AutoDock4 forcefield

While using the AutoDock4 forcefield, only the flex part of the receptor is necessary, as well as the affinity maps. Once the receptor (flex part `1fpu_receptor_flex.pdbqt`), ligand `1iep_ligand.pdbqt` and maps `1fpu_receptor_rigid` were prepared, you can perform the flexible side-chain docking by simply running the following command line:

```
$ vina --flex 1fpu_receptor_flex.pdbqt --ligand 1iep_ligand.pdbqt \
       --maps 1fpu_receptor_rigid --scoring ad4 \
       --exhaustiveness 32 --out 1fpu_ligand_flex_ad4_out.pdbqt
```

Running AutoDock Vina will write a PDBQT file called `1fpu_ligand_flex_ad4_out.pdbqt` contaning all the poses found during the molecular docking as well as the Thr315 sidechain conformations, and also present docking information to the terminal window.

4.b. Using Vina forcefield

As well as for the fully rigid molecular docking, you only need to specify the center and dimensions (in Angstrom) of the grid. Here, instead of specifying each parameters for the grid box using the arguments `--center_x`, `--center_y`, `--center_z` and `--size_x`, `--size_y`, `--size_z`, we will also store all those informations in a text file `1fpu_receptor_rigid_vina_box.txt`.

Listing 4: Content of the config file (**1fpu_receptor_rigid_vina_box.txt**) for AutoDock Vina

```
center_x = 15.190
center_y = 53.903
center_z = 16.917
size_x = 20.0
size_y = 20.0
size_z = 20.0
```

However, when using the Vina forcefield, you will need to specify both the rigid `1fpu_receptor_rigid.pdbqt` (needed to compute internally the affinity maps) and flex part `1fpu_receptor_flex.pdbqt` of the receptor. To perform the same docking experiment but using Vina forcefield run the following command line:

```
$ vina --receptor 1fpu_receptor_rigid.pdbqt --flex 1fpu_receptor_flex.pdbqt \
       --ligand 1iep_ligand.pdbqt --config 1fpu_receptor_rigid_vina_box.txt \
       --exhaustiveness 32 --out 1fpu_ligand_flex_vina_out.pdbqt
```

Tip: Alternatively, you can use the Vinardo forcefield by adding the `--scoring vinardo` option.

Running AutoDock Vina will write a PDBQT file called `1fpu_ligand_flex_vina_out.pdbqt`.

2.8.6 5. Results

Warning: Please don't forget that energy scores giving by the AutoDock and Vina forcefield are not comparable between each other.

5.a. Using AutoDock forcefield

The predicted free energy of binding should be about `-15 kcal/mol` for poses that are similar to the crystallographic pose.

```
Scoring function : ad4
Flex receptor: 1fpu_receptor_flex.pdbqt
Ligand: ../data/1iep_ligand.pdbqt
Exhaustiveness: 32
CPU: 0
Verbosity: 1

Reading AD4.2 maps ... done.
Performing docking (random seed: -1132104431) ...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
```

(continues on next page)

(continued from previous page)

```
*****************************************************

mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -15.41          0          0
   2       -14.95      1.164      1.803
   3       -13.92      1.112      1.744
   4       -13.39      3.975      6.038
   5       -13.08       1.48      2.166
   6       -12.13      3.877      11.74
   7       -12.13      5.806      9.094
   8       -11.89      1.251      1.971
   9       -11.55      2.804      10.81
```

5.b. Using Vina forcefield

Using the vina forcefield, you should obtain a similar output from Vina with the best score around `-12 kcal/mol`.

```
Scoring function : vina
Rigid receptor: 1fpu_receptor_rigid.pdbqt
Flex receptor: 1fpu_receptor_flex.pdbqt
Ligand: ../data/1iep_ligand.pdbqt
Center: X 15.19 Y 53.903 Z 16.917
Size: X 20 Y 20 Z 20
Grid space: 0.375
Exhaustiveness: 32
CPU: 0
Verbosity: 1

Computing Vina grid ... done.
Performing docking (random seed: 1973662971) ...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
*****************************************************

mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -12.17          0          0
   2       -11.41       3.23       12.1
   3       -11.22      1.512      2.137
   4       -11.19       4.07         12
   5       -10.64      3.833      11.99
   6        -10.2      2.537      12.12
   7       -9.547      2.493      12.26
   8       -9.367      2.476      12.41
   9       -9.051      3.809      11.72
```

2.9 Multiple ligands docking

Vina is now able to dock simultaneously multiple ligands. This functionality may find application in fragment based drug design, where small molecules that bind the same target can be grown or combined into larger compounds with potentially better affinity.

The protein PDE in complex with two inhibitors (pdb id: 5x72) was used as an example to demonstrate the ability of the AutoDock Vina to dock successfully multiple ligands. The two inhibitors in this structure are stereoisomers, and only the R-isomer is able to bind in a specific region of the pocket, while both the R- and S-isomers can bind to the second location.

Note: This tutorial requires a certain degree of familiarity with the command-line interface. Also, we assume that you installed the ADFR software suite as well as the meeko Python package.

2.9.1 Materials for this tutorial

For this tutorial, all the basic material are provided and can be found in the `AutoDock-Vina/example/mulitple_ligands_docking/data` directory (or on GitHub). If you ever feel lost, you can always take a look at the solution here: `AutoDock-Vina/example/mulitple_ligands_docking/solution`. All the Python scripts used here (except for `prepare_receptor` and `mk_prepare_ligand.py`) are located in the `AutoDock-Vina/example/autodock_scripts` directory, alternatively you can also find them here on GitHub.

2.9.2 1. Preparing the flexible receptor

Exactly like the *Basic docking* tutorial, the method requires a receptor coordinate file that includes all hydrogen atoms. The file `5x72_receptorH.pdb` is provided (see `data` directory located at `<autodock-vina_directory>/example/multiple_ligands_docking/`). This file contains the receptor coordinates taken from the PDB entry `5x72`. It was manually obtained by extracting the receptor coordinates (using an text editor) from the original PDB file `5x72.pdb` in the `data` directory, and the hydrogen atoms added using reduce.

```
$ prepare_receptor -r 5x72_receptorH.pdb -o 5x72_receptor.pdbqt
```

If you are not sure about this step, the output PDBQT file `5x72_receptor.pdbqt` is available in the `solution` directory.

2.9.3 2. Prepare ligands

Here, we will prepare two ligands instead of only one. We will start from the SDF files `5x72_ligand_p59.sdf` and `5x72_ligand_p69.sdf` located in the `data` directory. They were also obtained directly from the PDB here: 5x72 (see `Download instance Coordinates` link for the P59 and P69 molecules). Since the ligand files do not include the hydrogen atoms, we are going to automatically add them.

Warning: We strongly advice you against using PDB format for preparing small molecules, since it does not contain information about bond connections. Please don't forget to always check the protonation state of your molecules before docking. Your success can sometimes hang by just an hydrogen atom. ;-)

```
$ mk_prepare_ligand.py -i 5x72_ligand_p59.sdf -o 5x72_ligand_p59.pdbqt
$ mk_prepare_ligand.py -i 5x72_ligand_p69.sdf -o 5x72_ligand_p69.pdbqt
```

The output PDBQT `5x72_ligand_p59.pdbqt` and `5x72_ligand_p69.pdbqt` can be found in the `solution` directory.

2.9.4 3. (Optional) Generating affinity maps for AutoDock FF

As well as for the docking with a fully rigid receptor, we need to generate a GPF file to precalculate the affinity maps for each atom type. However, instead of using the full receptor, affinity maps will be calculated only for the rigid part of the receptor (`5x72_receptor.pdbqt`).

To prepare the GPF file for the rigid part of the receptor:

```
$ pythonsh <script_directory>/prepare_gpf.py -l 5x72_ligand_p59.pdbqt -r 5x72_receptor.
↪pdbqt \
          -p npts='80,64,64' -p gridcenter='-15 15 129' -o 5x72_receptor.gpf
```

This time we manually specified the center of the grid `-p gridcenter='-15 15 129'` as well as its size `-p npts='80,64,64'`.

Listing 5: Content of the grid parameter file (**5x72_receptor.gpf**) for the receptor (**5x72_receptor.pdbqt**)

```
npts 80 64 64                        # num.grid points in xyz
gridfld 5x72_receptor.maps.fld       # grid_data_file
spacing 0.375                        # spacing(A)
receptor_types A C NA OA N SA HD     # receptor atom types
ligand_types A C F OA N HD           # ligand atom types
receptor 5x72_receptor.pdbqt         # macromolecule
gridcenter -15.000 15.000 129.000    # xyz-coordinates or auto
smooth 0.5                           # store minimum energy w/in rad(A)
map 5x72_receptor.A.map              # atom-specific affinity map
map 5x72_receptor.C.map              # atom-specific affinity map
map 5x72_receptor.F.map              # atom-specific affinity map
map 5x72_receptor.OA.map             # atom-specific affinity map
map 5x72_receptor.N.map              # atom-specific affinity map
map 5x72_receptor.HD.map             # atom-specific affinity map
elecmap 5x72_receptor.e.map          # electrostatic potential map
dsolvmap 5x72_receptor.d.map             # desolvation potential map
dielectric -0.1465                   # <0, AD4 distance-dep.diel;>0, constant
```

Warning: You might have to manually edit the GPF file and add addtional atom types if the second ligand contains different atom types not present in the ligand used for creating the GPF file.

To execute `autogrid4` using `5x72_receptor.gpf`, run the folllowing command line:

```
$ autogrid4 -p 5x72_receptor.gpf -l 5x72_receptor_rigid.glg
```

You should obtain as well the following files:

```
1fpu_receptor.maps.fld       # grid data file
1fpu_receptor.*.map          # affinity maps for A, C, HD, NA, N, OA atom types
1fpu_receptor.d.map          # desolvation map
1fpu_receptor.e.map          # electrostatic map
```

2.9.5 4. Running AutoDock Vina

4.a. Using AutoDock4 forcefield

When using the AutoDock4 forcefield, you only need to provide the affinity maps and the ligand, while specifying that the forcefield used will be AutoDock4 using the option `--scoring ad4`.

```
$ vina --ligand 5x72_ligand_p59.pdbqt 5x72_ligand_p69.pdbqt --maps 5x72_receptor \
       --scoring ad4 --exhaustiveness 32 --out 5x72_ligand_ad4_out.pdbqt
```

4.b. Using Vina forcefield

As well as for the fully rigid molecular docking, you only need to specify the center and dimensions (in Angstrom) of the grid. Here, instead of specifying each parameters for the grid box using the arguments `--center_x`, `--center_y`, `--center_z` and `--size_x`, `--size_y`, `--size_z`, we will also store all those informations in a text file `5x72_receptor_vina_box.txt`.

Listing 6: Content of the config file (**5x72_receptor_vina_box.txt**) for AutoDock Vina

```
center_x = -15
center_y = 15
center_z = 129
size_x = 30
size_y = 24
size_z = 24
```

However, when using the Vina forcefield, you will need to specify the receptor `5x72_receptor.pdbqt` (needed to compute internally the affinity maps). To perform the same docking experiment but using Vina forcefield run the following command line:

```
$ vina --receptor 5x72_receptor.pdbqt --ligand 5x72_ligand_p59.pdbqt 5x72_ligand_p69.
↪pdbqt \
       --config 5x72_receptor_vina_box.txt \
       --exhaustiveness 32 --out 5x72_ligand_vina_out.pdbqt
```

Tip: Alternatively, you can use the Vinardo forcefield by adding the `--scoring vinardo` option.

Running AutoDock Vina will write a PDBQT file called `5x72_ligand_flex_vina_out.pdbqt`.

2.9.6 5. Results

Warning: Please don't forget that energy scores giving by the AutoDock and Vina forcefield are not comparable between each other.

5.a. Using AutoDock forcefield

The predicted free energy of binding should be about `-18 kcal/mol` for poses that are similar to the crystallographic pose. Using the AutoDock4 scoring function, the first two sets of poses (top 2) need to be considered to show also a

good overlap with the crystallographic poses

```
Scoring function : ad4
Ligands:
  - 5x72_ligand_p59.pdbqt
  - 5x72_ligand_p69.pdbqt
Exhaustiveness: 32
CPU: 0
Verbosity: 1

Reading AD4.2 maps ... done.
Performing docking (random seed: 1295744643) ...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************

mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -18.94          0          0
   2       -18.62      1.634      3.349
   3        -18.4      1.413      3.312
   4       -18.24      1.341      3.921
   5       -18.03      1.599      9.262
   6       -17.93      1.631      9.166
   7       -17.84      1.928      4.933
   8       -17.74       1.74      8.879
   9       -17.74          2      9.433
```

5.b. Using Vina forcefield

Using the vina forcefield, you should obtain a similar output from Vina with the best score around `-21 kcal/mol`. Using the Vina scoring function, the best set of poses (top 1) shows an excellent overlap with the crystallographic coordinates for one of the isomers.

```
Scoring function : vina
Rigid receptor: 5x72_receptor.pdbqt
Ligands:
  - 5x72_ligand_p59.pdbqt
  - 5x72_ligand_p69.pdbqt
Center: X -15 Y 15 Z 129
Size: X 30 Y 24 Z 24
Grid space: 0.375
Exhaustiveness: 32
CPU: 0
Verbosity: 1

Computing Vina grid ... done.
Performing docking (random seed: -2141167371) ...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************
```

(continues on next page)

(continued from previous page)

```
mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -21.32          0          0
   2       -20.94      1.061      3.648
   3       -20.73      1.392      3.181
   4       -19.93      1.744      4.841
   5       -19.34      1.384      3.352
   6       -19.05      1.185      9.184
   7        -18.9      1.198      3.586
   8       -18.76      1.862      8.986
   9       -18.63      1.749      9.194
```

2.10 Docking with zinc metalloproteins

Zinc is present in a wide variety of proteins and is important in the metabolism of most organisms. Zinc metalloenzymes are therapeutically relevant targets in diseases such as cancer, heart disease, bacterial infection, and Alzheimer's disease. In most cases a drug molecule targeting such enzymes establishes an interaction that coordinates with the zinc ion. Thus, accurate prediction of the interaction of ligands with zinc is an important aspect of computational docking and virtual screening against zinc containing proteins.

The AutoDock4 force field was extended to include a specialized potential describing the interactions of zinc-coordinating ligands. This potential describes both the energetic and geometric components of the interaction. The new force field, named AutoDock4Zn, was calibrated on a data set of 292 crystal complexes containing zinc. Redocking experiments show that the force field provides significant improvement in performance in both free energy of binding estimation as well as in root-mean-square deviation from the crystal structure pose.

Note: This tutorial requires a certain degree of familiarity with the command-line interface. Also, we assume that you installed the ADFR software suite as well as the meeko Python package.

Note: Please cite this paper if you are using this protocol in your work:

- Santos-Martins, D., Forli, S., Ramos, M. J., & Olson, A. J. (2014). AutoDock4Zn: an improved AutoDock force field for small-molecule docking to zinc metalloproteins. Journal of chemical information and modeling, 54(8), 2371-2379.

2.10.1 Materials for this tutorial

For this tutorial, all the basic material are provided and can be found in the `AutoDock-Vina/example/docking_with_zinc_metalloproteins/data` directory (or on GitHub). If you ever feel lost, you can always take a look at the solution here: `AutoDock-Vina/example/docking_with_zinc_metalloproteins/solution`. All the Python scripts used here (except for `prepare_receptor` and `mk_prepare_ligand.py`) are located in the `AutoDock-Vina/example/autodock_scripts` directory, alternatively you can also find them here on GitHub.

2.10.2 1. Preparing the receptor

During this step we will create the PDBQT file of the receptor using the PDB file called `proteinH.pdb`, containing all the hydrogen atoms, and add the tetrahedral zinc pseudo atoms (`TZ`) around the zinc ion. TZ atoms represent the preferred position for tetrahedral coordination by the ligand. This file contains the receptor coordinates of chain A and B taken from the PDB entry 1s63.

To prepare the receptor, execute the following command lines:

```
$ prepare_receptor -r protein.pdb -o protein.pdbqt -U nphs_lps_waters
$ pythonsh <script_directory>/zinc_pseudo.py -r protein.pdbqt -o protein_tz.pdbqt
```

NOTE: default for *prepare_receptor* is to remove chains consisting exclusively of non-standard residues (e.g. Zn), If zinc is in a chain without standard protein residues, option -U nphs_lps_waters` will prevent zinc removal. The execution of these two commands should output these two messages. One informing us that the charge for the zinc ion was not set by `prepare_receptor`. In this context, the message can be safely ignored since the ligand will interact preferentially with the zinc pseudo atoms (TZ). The PDBQT output files can be found in the `solution` directory.

```
Sorry, there are no Gasteiger parameters available for atom proteinH:B: ZN1001:ZN
```

The second message is telling us that only one zinc pseudo atom (TZ) was added to the receptor.

```
Wrote 1 TZ atoms on protein_tz.pdbqt.
```

2.10.3 2. Preparing the ligand

The second step consists to prepare the ligand, by converting the SDF file `1s63_ligand.msdf` to a PDBQT file readable by AutoDock Vina. As usual, we will use the `mk_prepare_ligand.py` Python script from `Meeko` (see installation instruction here: *Software requirements*) for this task. For your convenience, the molecule file `1s63_ligand.sdf` is provided (see `data` directory). But you can obtain it directly from the PDB here: 1s63 (see `Download instance Coordinates` link for the 778 molecule. Since the ligand file does not include the hydrogen atoms, we are going to automatically add them.

```
$ mk_prepare_ligand.py -i 1s63_ligand.sdf -o 1s63_ligand.pdbqt
```

The output PDBQT `1s63_ligand.pdbqt` can be found in the `solution` directory.

2.10.4 3. Generating affinity maps

The preparation script `prepare_gpf4zn.py` will be used to generate a special GPF file for docking with zinc pseudo atoms:

```
$ pythonsh <script_directory>/prepare_gpf4zn.py -l 1s63_ligand.pdbqt -r protein_tz.pdbqt␣
↪\
-o protein_tz.gpf  -p npts=40,30,50 -p gridcenter=18,134,-1 \
-p parameter_file=AD4Zn.dat
```

The `-p` flag is used to set the box center (`gridcenter`) and size (`npts`) along with the `parameter_file` specific for this case. After execution, you should obtain a GPF file called `protein_tz.gpf` containing this:

```
npts 40 30 50                        # num.grid points in xyz
parameter_file AD4Zn.dat             # force field default parameter file
gridfld protein_tz.maps.fld          # grid_data_file
```

(continues on next page)

(continued from previous page)

```
spacing 0.375                          # spacing(A)
receptor_types A C TZ NA ZN OA N P SA HD # receptor atom types
ligand_types A C Cl NA OA N HD         # ligand atom types
receptor protein_tz.pdbqt              # macromolecule
gridcenter 18 134 -1                   # xyz-coordinates or auto
smooth 0.5                             # store minimum energy w/in rad(A)
map protein_tz.A.map                   # atom-specific affinity map
map protein_tz.C.map                   # atom-specific affinity map
map protein_tz.Cl.map                  # atom-specific affinity map
map protein_tz.NA.map                  # atom-specific affinity map
map protein_tz.OA.map                  # atom-specific affinity map
map protein_tz.N.map                   # atom-specific affinity map
map protein_tz.HD.map                  # atom-specific affinity map
elecmap protein_tz.e.map               # electrostatic potential map
dsolvmap protein_tz.d.map                # desolvation potential map
dielectric -0.1465                     # <0, AD4 distance-dep.diel;>0, constant
nbp_r_eps 0.25 23.2135 12 6 NA TZ
nbp_r_eps 2.1   3.8453 12 6 OA Zn
nbp_r_eps 2.25  7.5914 12 6 SA Zn
nbp_r_eps 1.0   0.0    12 6 HD Zn
nbp_r_eps 2.0   0.0060 12 6 NA Zn
nbp_r_eps 2.0   0.2966 12 6  N Zn
```

The AutoDock4Zn forcefield is mostly defined by non bonded pairwise potentials which are written to the GPF file `protein_tz.gpf` in the form of `npb_r_eps` keywords. The file `AD4Zn.dat` includes the definition of the TZ atom type for the AutoDock forcefield. The keyword `parameter_file` in the GPF file specifies `AD4Zn.dat` as the forcefield to be used, so AutoGrid requires a local copy of it in the working directory. Alternatively, the keyword `parameter_file` in the GPF can point to the full or relative path where `AD4Zn.dat` is located.

Warning: The behavior of the `npb_r_eps` keyword changed between autogrid 4.2.6 and 4.2.7. Be sure that you are using the latest version (AutoGrid 4.2.7.x.2019-07-11) of autogrid4 shipped with the ADFR Suite.

```
$ autogrid4 -p protein_tz.gpf -l protein_tz.glg
```

At this stage, all forcefield information has been encoded in the affinity maps, and the remaining steps are the same as in the standard AutoDock protocol.

2.10.5 4. Running AutoDock Vina

4.a. Using AutoDock4 forcefield

When using the AutoDock4 forcefield, you only need to provide the affinity maps and the ligand, while specifying that the forcefield used will be AutoDock4 using the option `--scoring ad4`.

```
$ vina --ligand 1s63_ligand.pdbqt --maps protein_tz --scoring ad4 \
       --exhaustiveness 32 --out 1s63_ligand_ad4_out.pdbqt
```

2.10.6 5. Results

The predicted free energy of binding should be about `-13 kcal/mol` for the best pose and should corresponds to the crystallographic pose.

```
Scoring function : ad4
Ligand: 1s63_ligand.pdbqt
Exhaustiveness: 32
CPU: 0
Verbosity: 1

Reading AD4.2 maps ... done.
Performing docking (random seed: 1984557646) ...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************

mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1        -13.5          0          0
   2          -13      2.518      4.707
   3       -12.56      2.116      2.499
   4       -12.44      3.041      4.021
   5       -12.12      2.975      6.211
   6       -11.96      2.814      6.336
   7       -11.91      3.244      6.477
   8       -11.32      3.783      5.654
   9       -11.31      2.856      3.867
```

2.11 Hydrated docking

In physiological environments, proteins and other biological structures are surrounded by water molecules. When a small-molecule binds to a protein, it must displace most of the waters occupying the binding cavity. However, rarely are all water molecules displaced. Some waters can be so strongly bound and conserved among similar proteins that from a ligand-docking perspective they are considered a part of the target structure, altering the binding site topography.

Thus a method was developed that uses the existing version of AutoDock4 and now the new version AutoDock Vina 1.2.x but modifies the force field to model explicit bridging water molecules. In tests, this method has shown improvement in the prediction of bound conformations of small fragment molecules, such as those used in fragment-based drug discovery. The protocol can be summarized by those steps:

1. The ligand is decorated with an ensemble of water molecules (represented by dummy atoms), which may or may not then contribute to the intermolecular interactions.
2. A modified AutoGrid map is then used during docking, giving a favorable score when the water is well placed and omitting the water if it overlaps with the receptor.
3. Finally, docked results are analyzing and poses are rescored using only water molecules that were retained.

In this tutorial, we are going to dock a fragment-size ligand (nicotine) with explicit water molecules in the acetylcholine binding protein (AChBP) structure (PDB entry 1uw6). It was shown that the absence of water molecules could have a dramatic influence in docking performance leading to either inaccurate scoring and/or incorrect pose. With hydrated docking, fragment-sized ligands show a overall RMSD improvement.

Note: This tutorial requires a certain degree of familiarity with the command-line interface. Also, we assume that you installed the ADFR software suite as well as the meeko Python package.

Note: The materials present is this tutorial can be also found here: https://www.nature.com/articles/nprot.2016.051. If you are using this tutorial or this docking method for your work, you can cite the following papers:

- Forli, S., & Olson, A. J. (2012). A force field with discrete displaceable waters and desolvation entropy for hydrated ligand docking. Journal of medicinal chemistry, 55(2), 623-638.
- Forli, S., Huey, R., Pique, M. E., Sanner, M. F., Goodsell, D. S., & Olson, A. J. (2016). Computational protein–ligand docking and virtual drug screening with the AutoDock suite. Nature protocols, 11(5), 905-919.

2.11.1 Materials for this tutorial

For this tutorial, all the basic material are provided and can be found in the `AutoDock-Vina/example/hydrated_docking/data` directory (or on GitHub). If you ever feel lost, you can always take a look at the solution here: `AutoDock-Vina/example/hydrated_docking/solution`. All the Python scripts used here (except for `prepare_receptor` and `mk_prepare_ligand.py`) are located in the `AutoDock-Vina/example/autodock_scripts` directory, alternatively you can also find them here on GitHub.

2.11.2 1. Preparing the receptor

The receptor can be prepared using the method described earlier in the following tutorials: *Basic docking* or *Flexible docking* if one wants to incorporate some sidechain flexibility. The file `1uw6_receptorH.pdb` is provided (see `data` directory located at `<autodock-vina_directory>/example/hydrated_docking/`). This file contains the receptor coordinates of chain A and B taken from the PDB entry `1uw6`.

```
$ prepare_receptor -r 1uw6_receptorH.pdb -o 1uw6_receptor.pdbqt
```

The output PDBQT file `1uw6_receptor.pdbqt` is available in `solution` directory if necessary.

2.11.3 2. Preparing the ligand

For the hydrated docking, explicit water molecules (W atoms) must be added to the molecule. And for that, we will use `Meeko` (see installation instruction here: *Software requirements*). For convenience, the molecule file `1uw6_ligand.sdf` is provided (see `data` directory). But you can obtain it directly from the PDB here: 1uw6 (see `Download instance Coordinates` link for the NCT molecule (chain U [A])). Since the ligand file does not include the hydrogen atoms, we are going to automatically add them. The option `-w` is use to add explicit water molecule to the molecule.

Warning: We strongly advice you against using PDB format for preparing small molecules, since it does not contain information about bond connections. Please don't forget to always check the protonation state of your molecules before docking. Your success can sometimes hang by just an hydrogen atom. ;-)

```
$ mk_prepare_ligand.py -i 1uw6_ligand.sdf -o 1uw6_ligand.pdbqt -w
```

In total, 2 water molecules were added to the fragment. If you were not able to generate the `1uw6_ligand.pdbqt` file, you can look at the `solution` directory.

2.11.4 3. Generating affinity maps

As well as for the *Basic docking* or *Flexible docking* tutorials, we will also need to calculate the affinity maps for each atom types present in the ligand. However, this time we will also need to craft a special `W` affinity map for the water molecules attached to the ligand. This `W` affinity map is obtained by combining the `OA` and `HD` grid maps, therefore we are going to use the `-p ligand_types='A,C,OA,N,HD'` option to be sure those atom types are included in the GPF, in addition of the ligand atom types, while ignoring the `W` atom type:

```
$ pythonsh <script_directory>/prepare_gpf.py -l 1uw6_ligand.pdbqt -r 1uw6_receptor.pdbqt␣
↪-y \
           -p ligand_types='A,NA,C,HD,N,OA' \
```

The option `-y` specifies that we want to center automatically the grid around the ligand. After manually adding the `OA` atom type, you should have a GPF file called `1uw6_receptor.gpf` that looks like this:

Listing 7: Content of the grid parameter file (**1uw6_receptor.gpf**) for the receptor (**1uw6_receptor.pdbqt**)

```
npts 40 40 40                           # num.grid points in xyz
gridfld 1uw6_receptor.maps.fld          # grid_data_file
spacing 0.375                           # spacing(A)
receptor_types A C NA OA N SA HD        # receptor atom types
ligand_types A NA C HD N OA             # ligand atom types
receptor 1uw6_receptor.pdbqt            # macromolecule
gridcenter 83.640 69.684 -10.124        # xyz-coordinates or auto
smooth 0.5                              # store minimum energy w/in rad(A)
map 1uw6_receptor.A.map                 # atom-specific affinity map
map 1uw6_receptor.NA.map                # atom-specific affinity map
map 1uw6_receptor.C.map                 # atom-specific affinity map
map 1uw6_receptor.HD.map                # atom-specific affinity map * ADD OA IF NOT␣
↪PRESENT *
map 1uw6_receptor.N.map                 # atom-specific affinity map
map 1uw6_receptor.OA.map                # atom-specific affinity map * ADD OA IF NOT␣
↪PRESENT *
elecmap 1uw6_receptor.e.map             # electrostatic potential map
dsolvmap 1uw6_receptor.d.map                # desolvation potential map
dielectric -0.1465                      # <0, AD4 distance-dep.diel;>0, constant
```

You can now execute `autogrid4` using the GPF file called `1uw6_receptor.gpf` and generate the additional water map `W` by combining `OA` and `HD` affinity maps using `mapwater.py`:

```
$ autogrid4 -p 1uw6_receptor.gpf -l 1uw6_receptor.glg
$ python <script_directory>/mapwater.py -r 1uw6_receptor.pdbqt -s 1uw6_receptor.W.map
```

For more informations about the `mapwater.py` command tool and all the available options, just type `mapwater.py`. After executing this command, you should obtain a new affinity map called `1uw6_receptor.W.map` and the following the output:

```
ADD PWD AND FILE SUMMARY
  receptor :  1uw6_receptor.pdbqt
      OA map -> 1uw6_receptor.OA.map
      HD map -> 1uw6_receptor.HD.map
 => Water map weight : DEFAULT [ 0.60 ]
```

(continues on next page)

(continued from previous page)

```
 MapWater generator
=====================
 mode      :  BEST
 weight    :   0.6
 HD_weight :   1.0
 OA_weight :   1.0
 entropy   :   -0.2

    Output info
--------------------
 filename  : 1uw6_receptor.W.map
 OA points : 91.73%
 HD points : 8.27%

 lowest  map value : -0.99
 highest map value : -0.01
```

2.11.5 4. Running AutoDock Vina

4.a. Using AutoDock4 forcefield

Now that you generated the ligand with explicit water molecules attached (`1uw6_ligand.pdbqt`) and the extra affinity map for the `W` atom type (`1uw6_receptor.W.map`), you can do the molecular docking with Vina using the AutoDock4 forcefield:

```
$ vina  --ligand 1uw6_ligand.pdbqt --maps 1uw6_receptor --scoring ad4 \
        --exhaustiveness 32 --out 1uw6_ligand_ad4_out.pdbqt
```

4.b. Using Vina forcefield

Warning: While this method was calibrated and validated with the AutoDock4 forcefield, we strongly advice you against using this protocol with the Vina and Vinardo forcefield.

2.11.6 5. Results and post-processing

Warning: Be aware that with this implementation of the method, it is difficult to compare results obtained with very diverse ligands without doing extra of post-processing on the results, because the energy estimation needs to be normalized. For this reason, the method is not suitable for virtual screenings. This doesn't affect the structural accuracy, so comparisons within docking poses are fine. An improved scoring function to overcome this issue is in the works.

The predicted free energy of binding should be about `-8 kcal/mol` for poses that are similar to the crystallographic pose.

```
Scoring function : ad4
Ligand: 1uw6_ligand.pdbqt
```

(continues on next page)

(continued from previous page)

```
Exhaustiveness: 32
CPU: 0
Verbosity: 1

Reading AD4.2 maps ... done.
Performing docking (random seed: -655217817) ...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************

mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -8.077          0          0
   2        -7.63      2.038      2.684
   3       -7.382      2.378      2.747
   4        -7.27      2.063      2.538
   5       -7.138      1.861      5.391
   6       -7.129          2      2.542
   7       -7.078      3.307      5.442
   8       -7.065       2.22      4.872
   9       -7.051      3.135      5.636
```

Docking results are filtered by using the receptor to remove displaced waters and the W map file to rank the conserved ones as strong or weak water molecules.

```
$ python <script_directory>/dry.py -r 1uw6_receptor.pdbqt -m 1uw6_receptor.W.map -i 1uw6_
↪ligand_ad4_out.pdbqt
```

For more informations about the `dry.py` command tool and all the available options, just type `dry.py`. Running the previous command should give you this output:

```
                 ____
                /\  _`\
                \ \ \/\ \  _ __  __  __
                 \ \ \ \ \/\`'__\\ \/\ \
                  \ \ \_\ \ \ \/\ \ \_\ \
                   \ \____/\ \_\ \/`____ \
                    \/___/  \/_/  `/___/> \
                                     /\___/
                                     \/__/


========================== INPUT DATA ==========================
 importing ATOMS from  1uw6_ligand_ad4_out.pdbqt

 [ using map file 1uw6_receptor.W.map ]
================================================================


 receptor structure loaded           [ 4069 atoms ]
 receptor 5A shell extracted             [ 480 atoms in 5 A shell ]
```

(continues on next page)

(continued from previous page)

```
removing ligand/ligand overlapping waters    [ 5 water(s) removed ]
removing ligand/receptor overlapping waters      [ 8 water(s) removed ]

scanning grid map for conserved waters...    [ filtered pose contains 5 waters ]

water grid score results [ map: 1uw6_receptor.W.map ]
    [ Water STRONG ( -0.92 ) +++ ]
    [ Water STRONG ( -0.66 ) +++ ]
    [ Water  WEAK  ( -0.50 )  +  ]
    [ Water STRONG ( -0.83 ) +++ ]
    [ Water STRONG ( -0.99 ) +++ ]
```

Waters are ranked (STRONG, WEAK) and scored inside the output file `1uw6_ligand_ad4_out_DRY_SCORED.pdbqt` with the calculated energy.

2.12 Docking with macrocycles

Macrocycles are made flexible by default since Meeko v0.3.0. Thus, the basic docking tutorial suffices for docking macrocycles flexibly.

AutoDock Vina is not able to manage directly the flexibility associated with bonds in cyclic molecules. Different approaches can be used to dock macrocyclic molecules, like identifying one or more low energy conformations and docking them as different ligands, but generating them and docking them separately can be a time-consuming task. But AutoDock Vina (and AutoDock-GPU) has a specialized protocol to dock macrocycles while modeling their flexibility on-the-fly.

The current implementation is described in our paper on the D3R Grand Challenge 4 (see below) and is summarized herein:

1. One of the bonds in the ring structure is broken, resulting in an open form of the macrocycle that removes the need for correlated torsional variations, and enabling torsional degrees of freedom to be explored independently.
2. To each of the atoms of the broken bond, a dummy-atom is added. The distance between a dummy-atom and its parent atom is the same as the length of the broken bond, and the 1-3 angle also matches the original geometry.
3. During the docking, a linear attractive potential is applied on each dummy-atom to restore the bond resulting in the closed ring form. Thus, macrocycle conformations are sampled while adapting to the binding pocket, at the cost of increased search complexity with the added extra rotatable bonds.

Note: Even though dummy atoms are added and bonds are broken in the PDBQT representation of macrocycles, using Meeko's script mk_export.py (or class RDKitMolCreate from Python) will convert the docking output from PDBQT to MOL/SDF with correct connectivity and no dummy atoms.

The followings papers can be cited if this method is important for your publication:

- Holcomb, M., Santos-Martins, D., Tillack, A., & Forli, S. (2022). Performance evaluation of flexible macrocycle docking in AutoDock. QRB Discovery, 3, E18. doi:10.1017/qrd.2022.18
- Forli, S., & Botta, M. (2007). Lennard-Jones potential and dummy atom settings to overcome the AUTODOCK limitation in treating flexible ring systems. Journal of chemical information and modeling, 47(4), 1481-1492
- Santos-Martins, D., Eberhardt, J., Bianco, G., Solis-Vasquez, L., Ambrosio, F. A., Koch, A., & Forli, S. (2019). D3R Grand Challenge 4: prospective pose prediction of BACE1 ligands with AutoDock-GPU. Journal of Computer-Aided Molecular Design, 33(12), 1071-1081

2.13 Python scripting

2.13.1 Materials for this tutorial

For this tutorial, all the basic material are provided and can be found in the `AutoDock-Vina/example/python_scripting` directory (or on GitHub).

2.13.2 First example

Let's begin with a very simple example of an AutoDock Vina script. It loads affinity maps for the AutoDock forcefield with their filenames starting by *1iep*, then load a PDBQT file containing a ligand called *1iep_ligand.pdbqt*, score the current pose, minimize it, dock it and save the minimized pose to a PDBQT file called *1iep_ligand_minimized.pdbqt*. For the full documentation about the *Vina* Python package, see *Python documentation*.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
#
# My first example with AutoDock Vina in python
#

from vina import Vina


v = Vina(sf_name='vina')

v.set_receptor('1iep_receptor.pdbqt')

v.set_ligand_from_file('1iep_ligand.pdbqt')
v.compute_vina_maps(center=[15.190, 53.903, 16.917], box_size=[20, 20, 20])

# Score the current pose
energy = v.score()
print('Score before minimization: %.3f (kcal/mol)' % energy[0])

# Minimized locally the current pose
energy_minimized = v.optimize()
print('Score after minimization : %.3f (kcal/mol)' % energy_minimized[0])
v.write_pose('1iep_ligand_minimized.pdbqt', overwrite=True)

# Dock the ligand
v.dock(exhaustiveness=32, n_poses=20)
v.write_poses('1iep_ligand_vina_out.pdbqt', n_poses=5, overwrite=True)
```

You can find this script in the *example* folder of AutoDock-Vina available on Github. To execute it from a command line, go to your terminal/console/command prompt window. Navigate to the *examples* folder by typing

```
$ cd <examples_directory>/python_scripting
$ python first_example.py
```

Let's go through the script line by line and see how it works.

```
from vina import Vina
```

This line is just for Python to tell it to load the Python package `vina`.

```
v = Vina(sf_name='vina')
```

This line creates the `Vina` object that will be use for the subsequent taks. More precisely, it specifies also the forcefield that will be used for the molecular docking, here the Vina forcefield will be used. In the case, you want to use another forcefield like AutoDock4 or Vinardo, just replace `vina` by `ad4` or `vinardo`. Other options can be define, like the number of cpu that will be used during the molecule docking. By default, Vina will use the avaialble cpus on you machine (`cpu=0`), but you could use only one with `cpu=1`.

```
v.set_receptor('1iep_receptor.pdbqt')
```

Here, we are loading a PDBQT file called `1iep_receptor.pdbqt` containing the receptor. If necessary, PDBQT file containing the flexible sidechains can be also load at the same time by doing `v.set_receptor('1iep_rigid.pdbqt', '1iep_flex.pdbqt')`.

```
v.set_ligand_from_file('1iep_ligand.pdbqt')
v.compute_vina_maps(center=[15.190, 53.903, 16.917], box_size=[20, 20, 20])
```

The next lines are used to first load a PDBQT file containing the ligand called `1iep_ligand.pdbqt` and then compute the affinity maps for each ligand atom types accroding to the Vina forcefield. You might need to read first the tutorial *Basic docking* to learn how to create a PDBQT file of a ligand. There is a small subility here, the behavior of the `compute_vina_maps()` function changes if the ligand was loaded before or after computing the vina maps. If no ligand was initialized, `compute_vina_maps()` will compute the affinity map for each atom types defined in the Vina forcefield (22 in total). This is very useful when we want to dock ligands in batch (a.k.a virtual screening) but we don't necessarily know beforehand what atom types will be necessary for thoses ligands. Alternately to `set_ligand_from_file()`, you could also load a molecule using a molecule string in PDBQT format using the `set_ligand_from_string()` function.

```
# Score the current pose
energy = v.score()
print('Score before minimization: %.3f (kcal/mol)' % energy[0])
```

Next, we simply ask Vina to calculate the energy (*score*) of the current pose using the forcefield defined at the beginning, and retrieve the energy of each component in a numpy array. In this case, we are going to print to the screen only the total energy of the current pose. This task is often useful when you want to get the energy from the specific pose.

```
# Minimized locally the current pose
energy_minimized = v.optimize()
print('Score after minimization : %.3f (kcal/mol)' % energy_minimized[0])
v.write_pose('1iep_ligand_minimized.pdbqt', overwrite=True)
```

This line tells AutoDock Vina to perform a local energy minimization and show the total energy. It is useful sometimes to perform a quick energy minization after manually placing a ligand in a pocket and to remove possible steric clashes with itself and the receptor.

```
# Dock the ligand
v.dock(exhaustiveness=32, n_poses=20)
v.write_poses('1iep_ligand_vina_out.pdbqt', n_poses=5, overwrite=True)
```

Finally, we run the molecular docking. Here we will ask *Vina* to run 32 consecutive Monte-Carlo samplings using the `exhaustiveness` argument and store 20 poses (`n_poses`) during the search. At the end, we will write a PDBQT file called `1iep_ligand_vina_out.pdbqt` containing only the 5 first poses (`n_poses`), ranked by score. Of course, this can be change to 20 to include all the poses that were saved during the calculations, at the condition that the energy

difference between the best pose and the 20th pose if less than 3 kcal/mol. This behavior can be changed using the `energy_range` argument to an higher value.

2.14 AutoDock Vina

2.14.1 vina.vina

class vina.vina.**Vina**(*sf_name='vina'*, *cpu=0*, *seed=0*, *no_refine=False*, *verbosity=1*)

Bases: `object`

__init__(*sf_name='vina'*, *cpu=0*, *seed=0*, *no_refine=False*, *verbosity=1*)

Initialize a Vina object.

Parameters

- **sf_name** (*str*) – Scoring function name to use (Vina or ad4) (default: vina)
- **cpu** (*int*) – Number of CPU to use (default: 0; use all of them)
- **seed** (*int*) – Random seed (default: 0; ramdomly choosed)
- **no_refine** (*boolean*) – when receptor is provided, do not use explicit receptor atoms (instead of precalculated grids) for: (1) local optimization and scoring after docking, (2) –local_only jobs, and (3) –score_only jobs (default: False)
- **verbosity** (*int*) – verbosity 0: not output, 1: normal, 2: verbose (default: 1; some output)

__str__()

Print basic information about Docking configuration (rigid receptor, flex receptor, ligands, scoring function, weights, no_refine, box center, box dimensions, box spacing, box even nelements, seed).

cite()

Print citation message.

info()

Return information about the docking configuration.

Returns

Dictionary of information about the docking configuration.

Information:
rigid_receptor (str), flex_receptor (str), ligands (list), scoring_function (str), weights (tuple), no_refine (bool), box_center (list), box_size (list), box_spacing (float), box_even_elements (bool), seed (int)

Return type
dict (str)

set_receptor(*rigid_pdbqt_filename=None*, *flex_pdbqt_filename=None*)

Set receptor.

Parameters

- **rigid_pdbqt_filename** (*str*) – rigid pdbqt receptor filename (default: None)
- **flex_pdbqt_filename** (*str*) – flexible residues pdbqt filename (default: None)

set_ligand_from_file(*pdbqt_filename*)

Set ligand(s) from a file. The chemical file format must be PDBQT.

Parameters
pdbqt_filename (*str or list*) – Name or list of PDBQT filename(s)

set_ligand_from_string(*pdbqt_string*)

Set ligand(s) from a string. The chemical file format must be PDBQT.

Parameters
pdbqt_string (*str or list*) – string or list of PDBQT strings

set_weights(*weights*)

Set potential weights for vina or ad4 scoring function.

Parameters
weights (*list*) – list or weights

compute_vina_maps(*center*, *box_size*, *spacing=0.375*, *force_even_voxels=False*)

Compute affinity maps using Vina scoring function.

Parameters

- **center** (*list*) – center position
- **box_size** (*list*) – size of the box in Angstrom
- **spacing** (*float*) – grid spacing (default: 0.375)
- **force_even_voxels** (*boolean*) – Force the number of voxels (NPTS/NELEMENTS) to be an even number (and forcing the number of grid points to be odd) (default: False)

load_maps(*map_prefix_filename*)

Load vina or ad4 affinity maps.

Parameters
map_prefix_filename (*str*) – affinity map prefix filename

write_maps(*map_prefix_filename='receptor'*, *gpf_filename='NULL'*, *fld_filename='NULL'*, *receptor_filename='NULL'*, *overwrite=False*)

Write affinity maps.

Parameters

- **map_prefix_filename** (*str*) – affinity map pathname (path directory + prefix)
- **gpf_filename** (*str*) – grid protein filename (default: NULL)
- **fld_filename** (*str*) – fld filename (default: NULL)
- **filename** (*receptor*) – receptor filename (default: NULL)
- **overwrite** (*bool*) – allow overwriting (default: false)

write_pose(*pdbqt_filename*, *remarks=''*, *overwrite=False*)

Write pose (after randomize or optimize).

Parameters

- **pdbqt_filename** (*str*) – output PDBQT filename
- **remarks** (*str*) – REMARKS to add in the PDBQT filename
- **overwrite** (*bool*) – allow overwriting (default: false)

write_poses(*pdbqt_filename*, *n_poses=9*, *energy_range=3.0*, *overwrite=False*)

Write poses from docking.

Parameters

- **pdbqt_filename** (*str*) – PDBQT file containing poses found
- **n_pose** (*int*) – number of poses to write (default: 9)
- **energy_range** (*float*) – maximum energy difference from best pose (default: 3.0 kcal/mol)
- **overwrite** (*bool*) – allow overwriting (default: false)

poses(*n_poses=9*, *energy_range=3.0*, *coordinates_only=False*)

Get poses from docking.

Parameters

- **n_pose** (*int*) – number of poses to retrieve (default: 9)
- **energy_range** (*float*) – maximum energy difference from best pose (default: 3.0 kcal/mol)
- **coordinates_only** (*bool*) – return coordinates for each pose only

Returns

PDBQT file string or Array of coordinates of poses if coordinates_only=True

Return type

str/ndarray

energies(*n_poses=9*, *energy_range=3.0*)

Get pose energies from docking.

Parameters

- **n_pose** (*int*) – number of poses to retrieve (default: 9)
- **energy_range** (*float*) – maximum energy difference from best pose (default: 3.0 kcal/mol)

Returns

Array of energies from each pose (rows=poses, columns=energies)

Vina/Vinardo FF:
columns=[total, inter, intra, torsions, intra best pose]

AutoDock FF:
columns=[total, inter, intra, torsions, -intra]

Return type

ndarray

randomize(*max_steps=10000*)

Randomize the input ligand conformation.

Parameters

max_steps (*int*) – Number of poses to generate for selection of the best one.

score(*unbound_energy=None*)

Score current pose.

Parameters
unbound_energy (*float*) – Optionally pass the unbound systems energy of the ligand.

Returns

Array of energies from current pose.

Vina/Vinardo FF:
columns=[total, lig_inter, flex_inter, other_inter, flex_intra, lig_intra, torsions, lig_intra best pose]

AutoDock FF:
columns=[total, lig_inter, flex_inter, other_inter, flex_intra, lig_intra, torsions, -lig_intra]

Return type
nadarray

optimize(*max_steps=0*)

Quick local BFGS energy optimization.

Parameters
max_steps (*int*) – Maximum number of local minimization steps (default: 0). When max_steps is equal to 0, the maximum number of steps will be equal to (25 + num_movable_atoms) / 3).

Returns

Array of energies from optimized pose.

Vina/Vinardo FF:
columns=[total, lig_inter, flex_inter, other_inter, flex_intra, lig_intra, torsions, lig_intra best pose]

AutoDock FF:
columns=[total, lig_inter, flex_inter, other_inter, flex_intra, lig_intra, torsions, -lig_intra]

Return type
nadarray

dock(*exhaustiveness=8*, *n_poses=20*, *min_rmsd=1.0*, *max_evals=0*)

Docking: global search optimization.

Parameters

- **exhaustiveness** (*int*) – Number of MC run (default: 8)
- **n_poses** (*int*) – number of pose to generate (default: 20)
- **min_rmsd** (*float*) – minimum RMSD difference between poses (default: 1.0 Ansgtrom)
- **max_evals** (*int*) – Maximum number of evaluation (default: 0; use heuristics rules)

2.14.2 vina.utils

vina.utils.**check_file_writable**(*fnm*)

Source: https://www.novixys.com/blog/python-check-file-can-read-write/

PYTHON MODULE INDEX

V

INDEX

Symbols

C

D

E

I

L

M

O

P

R

S

V

W