**DOCUMENTATION**

**Tutorial**

1. **Installing dependencies**

Follow the instructions detailed in the file *Installation\_Ubuntu.txt* in order to install the dependencies needed by VSpipe: MGLTools, OpenBabel, and Anaconda. Once you have them installed, you are ready to run VSpipe!

**OpenBabel, AutoDock scripts, and in-house python scripts**

**1. Open Babel in VSpipe**

Open Babel is a chemical toolbox designed to speak the many languages of chemical data. It can search, convert, analyse, or store data from molecular modelling, chemistry, solid-state materials, biochemistry, or other related area.

The following Open Babel options were used in VSpipe:  
  
1.1. Format conversion

The option -O was used to convert the ligands files into SDF format, which is readable by AutoDock.

1.2. Addition of ligand properties

Some ligands files may not contain all the properties that are important to benchmark the results of a VS. Open Babel tackles this issue with the option --add, giving the users the chance to specify after this flag the properties wanted to be added to the molecule of interest.

VSpipe uses this option to include for each ligand (i) canonical SMILES (cansmi), (ii) the number of hydrogen bond acceptors (HBA2), (iii) the number of hydrogen bond donors (HBD), (iv) the octanol/water partition coefficient (logP), (v) the topological polar surface area (TPSA), and (vi) the molecular weight (MW). Additionally, VSpipe uses the -b option to convert dative bonds, e.g. from [N+]([O-])=O to N(=O)=O, and the --unique option to avoid the presence of duplicates.

1.3. Conformers generation

The quality of the generated conformers will always affect the performance of the VS. Therefore, there are a wide range of options in Open Babel when the option --conformer is flagged that can be used to generate conformers according to specific parameters. VSpipe generates 3D conformers, but restricts them to 50 by using the options --nconf 50 and --gen3d, respectively. The hydrogens are deleted from the conformers by specifying the option -d because they are later added by the AutoDock scripts when preparing the conformers for the docking.

This avoids a bias in the positioning of the hydrogens as the conformers do not have hydrogens when they start the preparation step under the AutoDock scripts. The ligand energy was the metric chosen to score the conformers, hence we included the option --score energy. Last, we used the flag -m so a unique file with all the conformers was generated to easily proceed in the next steps of the VS.

**2. AutoDock scripts in VSpipe**

AutoDock is a suite of automated tools needed to carry out a VS. It includes different Python scripts that can be used in different steps, such as to prepare the receptor protein and the ligands or to summarise the docking results. Furthermore, it includes two freely available docking softwares: AD4 and Vina. AutoDock is part of the MGLTools package, although the docking softwares can also be individually downloaded from the AutoDock website.

2.1. Preparation of the target protein

The target protein may contain water molecules, metal ions, co-factors, heavy atoms, and could even have missing chains and/or side-chains; which would prevent a reliable docking. Additionally, it can sometimes include a co-crystallized ligand (or ligands) or more than one chain (protein domain), thus being crucial to select the correct chain of interest for the docking as well as to delete existing ligands. Therefore, the AutoDock script *prepare\_receptor4.py* is used in VSpipe to compute these preparation steps.

Particularly, the options chosen when this script is called through the pipeline code are related to the target protein charges, hydrogens, and waters. First of all, hydrogens are added (option -A hydrogens) without a bias in their positioning as the original hydrogens are removed in previous steps (see 1.3). Afterwards, Gasteiger charges are added and later merged by default while the non-polar hydrogens ADT assumes that non-polar hydrogens are hydrogens bonded to carbon atoms) are removed (option -U nphs). Finally, the water molecules are also removed as their presence in the active site could conflict with the docking process (option -U water).

2.2. Preparation of the ligands

The ligands also need a preparation before the docking takes place. The AutoDock python script used in VSpipe for this purpose is the *prepare\_ligand4.py*. This carries out the same tasks that those mentioned above to compute with the target proteins.

2.3. Scripts used to run AD4

*2.3.1. Grid parameter file and pre-calculation of the grid maps of interaction energies for different atom types*

AutoGrid is the program used to pre-calculate the grid map files containing the energy information of the specific atoms (such as aromatic carbons, aliphatic carbons, or hydrogen bonding oxygens) found in the target protein. This generates a grid log file (*GLG* file) for every ligand, the grid map files for every atom type found in the ligand, a desolvation map, an electrostatic map, and the grid size and grid field files. The users specify the grid parameter (*GPF* file) file that will be used by AutoGrid (*autogrid4* executable) setting the option -p before the pathway to the file. This *GPF* file can be manually created by the expert users with ADT or by using the python *prepare\_gpf4.py* script. VSpipe includes a sample of this *GPF* file, the *sample.gpf* (section 3.8), which can be easily modified by the users according to the target protein used in their VS. Note that this *sample.gpf* is one of the input files asked by VSpipe to be uploaded.

*2.3.2. Preparation of the docking parameter file and use of AD4*

AD4 is one of the softwares in the AutoDock suite used to perform the docking step in the VS. This program uses the grid map files previously calculated by AutoGrid and the docking parameter file (*DPF* file). In order to prepare it, the AutoDock script *prepare\_dpf4.py* can be used. VSpipe automatically creates this *DPF* file and calls it *sample.dpf* (sections 3.9 and 3.10), which contains all the information needed to correctly proceed with the VS without having to manually create this file. However, the users can always check the file and modify it, if necessary, before continuing with the VS. Once all the files are created, AD4 can be executed, which will generate the docking log files (*DLG* files) files.

*2.3.3. Summarising the docking results with AD4*

The python *summarize\_results4.py* script creates a text file summarizing the results found in the *DLG* files created by *autodock4* and saved in a specific directory. One summary file is output per ligand, *summary\_<ligand>.txt*.

2.4. Scripts used to run Vina

Vina uses the ILS global optimizer with its BFGS local optimization algorithm, instead of the LGA algorithm that AD4 does, to dock the ligands on the receptor. Furthermore, the scoring function used is not the same neither. Vina does not use the *autogrid4* program to pre-calculate the grid map files because it calculates them internally, thus the docking with Vina takes less time than with AD4, if AD4 is not parallelised.

VSpipe uses the two executable scripts needed to run Vina: *vina* and *vina\_split*. The first one sets the configuration file options and the location of the log file.

The configuration file must have the pathway to the target protein file and the grid size and the grid centre measures (in Angstroms). The second one is used to obtain separate files for every ligand used during the docking process (useful if more than one ligand is screened).

**3. In-house python scripts for VSpipe**

In addition to the Open Babel options and AutoDock scripts previously described, VSpipe uses other Python scripts we wrote to keep the workflow of the pipeline.

3.1. Processing the target protein – *clean\_protein.py*

This script generates a file with the first chain of the target protein. Therefore, the output file only has the information given in the lines of the input file that start with “ATOM”.

3.2. Addition of a metal ion for metalloproteinases – *adding\_metal\_ion.py*

This script offers the users the possibility to keep in the target protein the metal ion found in the active site of this protein. This is meant to be used only with metalloproteinases, as they may have the metal ion involved in the protein function.

3.3. Addition of the metal ion charge for metalloproteinases – *adding\_metal\_charge.py*

Only if the users decides to keep the metal ion in the target protein, this script is used together with *adding\_metal\_ion.py* by VSpipe. The script *adding\_metal\_charge.py* adds the charge of the metal ion kept in the target protein file in the correct position (AutoDock format).

3.4. Correction of the PDBQT target protein output file – *receptor\_pdbqt\_correction.py*

Sometimes, the target protein in the PDBQT format output by the *prepare\_receptor4.py* does not have the lines written in a correct position. In this case, this script shifts them to the correct position.

3.5. Addition of code names to the ligand or ligands – *to\_sdf\_correction.py*

If it is necessary, this script adds the ID code in the ligands files following the SDF format.

3.6. Deletion of atoms AutoDock cannot recognise – *atom\_deletion.py*

Specifically, the canonical smiles of the ligands are checked in order to find any non-supportive atoms by AutoDock. In this case, the script will remove them from the file.

3.7. Renaming the ligand or ligands with the correct IDs – *pdbs\_rename.py*

After using Open Babel to convert the ligands into the SDF format, the script renames them with their correct IDs.

3.8. Correction of the *sample.gpf* regarding the atom types – *generating\_correct\_sample\_gpf.py*

When the *sample.gpf* (section 2.3.1) file is used by AutoGrid to create the grid map files, sometimes the target protein atom types related are not the same as those find in the prepared target protein PDBQT file generated by AutoDock scripts (section 2.1). Therefore, this script corrects the *sample.gpf* file in order to have the correct atom types found in the prepared target protein.

3.9. Automatic generation of the docking parameter file – *generating\_correct\_dpf.py*

This script automatically generates the *sample.dpf* (section 2.3.2) without the need of manually moving it to a specific directory .

3.10. Rewriting correctly the docking parameter file – *dpf\_rewrite.py*

This script changes the path of the ligands prepared by AutoDock scripts (in PDBQT format) and the path of the *sample.gpf* (section 2.3.1) in order to correct the information found in the *sample.dpf* (section 2.3.2).

3.11. Creation of the *etc* directory with the *docking.list* file inside it to run the MPI version of AD4 – *creating\_etc\_dir.py*

This script is used to automatically create the *etc* directory and the *docking.list* file, which contains the names of the ligands that are going to be used during the docking process. Note that this script is only run by VSpipe if the MPI version of AD4 is used.

3.12. Creation of the output file – *datasheet.py*

This script is used to have the summary of the VS in two output files: *output.csv* and *output.tsv.* They contain the correct number of cells and all the information regarding the chemical properties calculated during the VS.

3.13. Configuration file to run Vina – *vina\_sample.py*

The configuration file needed for the *vina* executable to work is automatically created with this script (section 2.4).

3.14. Extraction of the output information – *values\_extraction\_AD4.py* and *values\_extraction\_vina.py*

The parameters calculated during the VS such as the ligand efficiency or ΔG are added in the output files (*output.csv* and *output.tsv*), previously generated by *datasheet.py* (section 3.12).

3.15. Filtering the output file according to the users’ decision – *filtering.py*

The output file *output.csv* is used for the filtering step by the *filtering.py* script. The output file can be filtered by any of the parameters used to rank the ligands and by any threshold required by the users with this script.