NAME

VinaPerformDocking.py - Perform docking

SYNOPSIS

VinaPerformDocking.py [--energyComponents <yes or no>] [--energyComponentsLabels <Label1, label2, Label3>] [--energyLabel <text>] [--energyRange <number>] [--exhaustiveness <number>] [--forcefield <AD4, Vina, or Vinardo>] [--forcefieldWeightParams <Name,Value,...>] [--gridSpacing <number>] [--gridSize <xsize,ysize,zsize>] [--infileParams <Name,Value,...>] [--maxEvaluations <number>] [--mergeHydrogens <yes or no>] [--minRMSD <number>] [--mode <Dock, LocalOptimizationOnly, ScoreOnly>] [--mp <yes or no>] [--mpParams <Name, Value,...>] [--numPoses <number>] [--numThreads <number>] [--outfileParams <Name,Value,...>] [--overwrite] [--precision <number>] [--quiet <yes or no>] [--randomSeed <number>] [--receptorFlexFile <receptor flex file>] [--skipRefinement <yes or no>] [--validateMolecules <yes or no>] [--vinaVerbosity <number>] [-w <dir>] -g <RefLigandFile or x,y,z> -r <receptorfile or maps prefix> -i <infile> -o <outfile>

VinaPerformDocking.py -h | --help | -e | --examples

DESCRIPTION

Dock molecules against a protein receptor using AutoDock Vina [Ref 168, 169]. The molecules must have 3D coordinates in input file. In addition, the hydrogens must be present for all molecules in input file.

No protein receptor preparation is performed during docking. It must be prepared employing standalone scripts available as part of AutoDock Vina. You may optionally specify flexible residues in the binding pocket to prepare a flexible receptor file and employ it for docking molecules along with the fixed receptor file.

The following three forecefileds are available to score molecules: AD4 (AutoDock4), Vina, and Vinardo (Vina RaDii Optimized) [Ref 170].

The supported input file formats are shown below:

```
Rigid/Flexible protein receptor files  - PDBQT(.pdbqt)
Reference ligand file: PDB(.pdb), Mol (.mol), SD (.sdf, .sd)

Input molecules file - Mol (.mol), SD (.sdf, .sd)
```

The supported output file format is: SD (.sdf, .sd).

The following output files are generated:

```
<OutfileRoot>.<OutfileExt> - Docked/scored molecules
<OutfileRoot>_Flex_Receptor.<OutfileExt> - Docked poses for flexible
    residues
```

The flexible receptor output file contains docked poses corresponding to flexible residues. It is only generated during 'Dock' value of '-m, --mode' option. The number of poses in this file matches those written to the output file containing docked molecules.

OPTIONS

--energyComponents <yes or no> [default: no]

Write out binding energy components of the total binding energy docking score to outfile. The following three energy components are written to outfile: intermolecula energy, internal energy, and torsions energy.

--energyComponentsLabels <Label1, label2, Label3> [default: auto]

A triplet of comma delimited values corresponding to energy data field labels for writing out the binding energy components to outfile. You must specify all three values. A value of 'None' implies the use of the default labels as shown below:

```
Label1: <ForcefieldName>_Intermolecular_Energy (kcal/mol)
Label2: <ForcefieldName>_Internal_Energy (kcal/mol)
Label3: <ForcefieldName>_Torsions_Energy (kcal/mol)
```

--energyLabel <text> [default: auto]

Energy data field label for writing out binding energy docking score to output file. Default: <ForcefieldName>_Total_Energy (kcal/mol).

--energyRange <number> [default: 3.0]

Maximum energy difference from the best pose during the generation of poses. Units: kcal/mol.

-e, --examples

Print examples.

--exhaustiveness <number> [default: 8]

Exhaustiveness of global MC search. The higher values make the search more exhaustive and it takes longer to complete. You may want to use '16' or '32' as the value of '--exhaustiveness ' to increase the accuracy of your pose prediction.

-f, --forcefield <AD4, Vina, or Vinardo> [default: Vina]

Forcefield to use for scoring. Possible values: AD4 (AutoDock 4), Vina [Ref 169, 169], or Vinardo (Vina RaDii Optimized) [Ref 170].

You must specify affinity maps using '-r, --receptor' option during the use of 'AD4' forcefield.

--forcefieldWeightParams <Name,Value,...> [default: auto]

A comma delimited list of parameter name and value pairs for forcefield scoring.

The supported parameter names along with their default values are are shown below for different forcefields:

```
AD4 (6 weights):

ad4Vdw, 0.1662, ad4HydrogenBond, 0.1209, ad4Electrostatic, 0.1406,
ad4Desolvation, 0.1322, ad4GlueLinearAttraction, 50.0,
ad4Rot, 0.2983

Vina (7 weights):

vinaGaussian1, -0.035579, vinaGaussian2, -0.005156,
vinaRepulsion, 0.840245, vinaHydrophobic, -0.035069,
vinaHydrogenBond, -0.587439, vinaGlueLinearAttraction, 50.0,
vinaRot, 0.05846

Vinardo (6 weights):

vinardoGaussian1, -0.045, vinardoRepulsion, 0.8,
vinardoHydrophobic, -0.035, vinardoHydrogenBond, -0.600
vinardoGlueLinearAttraction, 50.0, vinardoRot, 0.05846
```

The glue weight parameter corresponds to linear attraction for macrocycle closure and has the same value for AD4, Vina, and Vinardo. The rot weight has the same value for Vina and Vinardo.

-g, --gridCenter <RefLigandFile or x,y,z>

Reference ligand file for calculating the docking grid center or a triplet of comma delimited values in Angstrom corresponding to grid center.

This is required option. However, it is ignored during the specification of maps prefix for '-r, --receptor' option.

--gridSize <xsize,ysize,zsize> [default: 25.0, 25.0, 25.0]

Docking grid size in Angstrom.

--gridSpacing <number> [default: 0.375]

Docking grid spacing in Angstrom.

-h, --help

Print this help message.

-i, --infile <infile>

> Input file name containing molecules for docking against a receptor. The molecules must have 3D coordinates in input file. In addition, the hydrogens must be present for all molecules in input file. The input file may contain 3D conformers.

--infileParams <Name,Value,...> [default: auto]

> A comma delimited list of parameter name and value pairs for reading molecules from files. The supported parameter names for different file formats, along with their default values, are shown below:
>
> ```
> SD, MOL: removeHydrogens,no,sanitize,yes,strictParsing,yes
> ```

--maxEvaluations <number> [default: 0]

> Maximum number of evaluations to perform for each MC run during docking. By default, its value is set 0 and the number of MC evaluations is determined using heuristic rules.

--mergeHydrogens <yes or no> [default: auto]

> Merge hydrogens during preparation of molecules for docking. The hydrogens are automatically merged during 'AD4' value of '-f, --forcefield' option and its value is set to 'yes'. Otherwise, it's set to 'no'.

--minRMSD <number> [default: 1.0]

> Minimum RMSD between output poses in Angstrom.

-m, --mode <Dock, LocalOptimizationOnly, ScoreOnly> [default: Dock]

> Dock molecules or simply score molecules without performing any docking. The supported values along with a brief explanation of the expected behavior are shown below:
>
> ```
> Dock: Global search along with local optimization and scoring after
>     docking
> LocalOptimizationOnly: Local optimization and scoring without any docking
> ScoreOnly: Scoring without any local optimizatiom and docking
> ```
>
> The 'ScoreOnly" allows you to score 3D moleculed from input file which are already positioned in a binding pocket of a receptor.

--mp <yes or no> [default: no]

> Use multiprocessing.
>
> By default, input data is retrieved in a lazy manner via mp.Pool.imap() function employing lazy RDKit data iterable. This allows processing of arbitrary large data sets without any additional requirements memory.
>
> All input data may be optionally loaded into memory by mp.Pool.map() before starting worker processes in a process pool by setting the value of 'inputDataMode' to 'InMemory' in '--mpParams' option.
>
> A word to the wise: The default 'chunkSize' value of 1 during 'Lazy' input data mode may adversely impact the performance. The '--mpParams' section provides additional information to tune the value of 'chunkSize'.

--mpParams <Name,Value,...> [default: auto]

> A comma delimited list of parameter name and value pairs to configure multiprocessing.
>
> The supported parameter names along with their default and possible values are shown below:
>
> ```
> chunkSize, auto
> inputDataMode, Lazy   [ Possible values: InMemory or Lazy ]
> numProcesses, auto    [ Default: mp.cpu_count() ]
> ```
>
> These parameters are used by the following functions to configure and control the behavior of multiprocessing: mp.Pool(), mp.Pool.map(), and mp.Pool.imap().
>
> The chunkSize determines chunks of input data passed to each worker process in a process pool by mp.Pool.map() and mp.Pool.imap() functions. The default value of chunkSize is dependent on the value of 'inputDataMode'.
>
> The mp.Pool.map() function, invoked during 'InMemory' input data mode, automatically converts RDKit data iterable into a list, loads all data into memory, and calculates the default chunkSize using the following method as shown in its code:
>
> ```
> chunkSize, extra = divmod(len(dataIterable), len(numProcesses) * 4)
> ```

```
if extra: chunkSize += 1
```

For example, the default chunkSize will be 7 for a pool of 4 worker processes and 100 data items.

The mp.Pool.imap() function, invoked during 'Lazy' input data mode, employs 'lazy' RDKit data iterable to retrieve data as needed, without loading all the data into memory. Consequently, the size of input data is not known a priori. It's not possible to estimate an optimal value for the chunkSize. The default chunkSize is set to 1.

The default value for the chunkSize during 'Lazy' data mode may adversely impact the performance due to the overhead associated with exchanging small chunks of data. It is generally a good idea to explicitly set chunkSize to a larger value during 'Lazy' input data mode, based on the size of your input data and number of processes in the process pool.

The mp.Pool.map() function waits for all worker processes to process all the data and return the results. The mp.Pool.imap() function, however, returns the the results obtained from worker processes as soon as the results become available for specified chunks of data.

The order of data in the results returned by both mp.Pool.map() and mp.Pool.imap() functions always corresponds to the input data.

-n, --numPoses <number> [default: 1]

Number of docked poses to generate for each molecule and write out to output file. This option is only valid for 'Dock' value of '-m, --mode' option.

--numThreads <number> [default: auto]

Number of threads/CPUs to use for MC search calculation in Vina. The default value is set to 1 during multiprocessing for 'Yes' value of '--mp' option. Otherwise, it's set to 0 and rely on Vina detect and use all available CPUs for multi-threading.

-o, --outfile <outfile>

Output file name for writing out molecules. The flexible receptor residues are written to <OutfileRoot>_Flex_Receptor.<OutfileExt>.

--outfileParams <Name,Value,...> [default: auto]

A comma delimited list of parameter name and value pairs for writing molecules to files. The supported parameter names for different file formats, along with their default values, are shown below:

```
SD: kekulize,yes,forceV3000,no
```

--overwrite

Overwrite existing files.

--precision <number> [default: 2]

Floating point precision for writing energy values.

-q, --quiet <yes or no> [default: no]

Use quiet mode. The warning and information messages will not be printed.

--randomSeed <number> [default: 0]

Random seed for MC search calculations. A value of zero implies it's randomly chosen by Vina during the calculation.

-r, --receptor <receptor file or maps prefix>

Protein receptor file name or prefix for affinity map files corresponding to the fixed portion of the receptor.

You must specify affinity map files for 'AD4' forcefield. The affinity map files correspond to <MapsPrefix>.*.map and must be present

The supported receptor file format is PDBQT (.pdbqt). It must contain a prepared protein receptor ready for docking. You may prepare a PDBQT receptor file from a PDB file employing the command line scripts available with AutoDock Vina and Meeko. For example: prepare_receptor, prepare_flexreceptor.py, or or mk_make_recepror.py.

You may want to perform the following steps to clean up your PDB file before generating a PDBQT receptor file: Remove extraneous molecules such as solvents, ions, and ligand etc.; Extract data for a chain

containing the binding pocket of interest.

--receptorFlexFile <receptor flex file> [default: none]

Protein receptor file name corresponding to the flexible portion of the receptor. The supported receptor file format is PDBQT (.pdbqt). It must contain a prepared protein receptor ready for docking. You may prepare a flexible PDBQT receptor file from a PDB file employing the command line script prepare_flexreceptor or mk_make_recepror.py available with Autodock Vina and Meeko.

--skipRefinement <yes or no> [default: no]

Skip refinement. Vina is initialized to skip the use of explicit receptor atoms, instead of precalculated grids, during the following three docking modes: Dock, LocalOptimizationOnly, and ScoreOnly.

-v, --validateMolecules <yes or no> [default: yes]

Validate molecules for docking. The input molecules must have 3D coordinates and the hydrogens must be present. You may skip validation of molecules in input file containing all valid molecules.

--vinaVerbosity <number> [default: auto]

Verbosity level for running Vina. Possible values: 0 - No output; 1 - Normal output; 2 - Verbose. Default: 0 during multiprocessing for 'Yes' value of '--mp' option; otherwise, its set to 1. A non-zero value is not recommended during multiprocessing. It doesn't work due to the mingling of the Vina output from multiple processes.

-w, --workingdir <dir>

Location of working directory which defaults to the current directory.

EXAMPLES

To dock molecules using Vina forcefield against a prepared receptor corresponding to chain A extracted from PDB file 1R4L.pdb, multi-threading across all available CPUs during Vina MC search, calculating grid center form a reference ligand file, using grid box size of 25 Angstrom, generating one pose for each molecule, and write out a SD file containing docked molecules, type:

```
% VinaPerformDocking.py -g SampleACE2RefLigand.pdb
  -r SampleACE2Receptor.pdbqt -i SampleACE2Ligands.sdf
  -o SampleACE2LigandsOut.sdf
```

To run the first example for generating multiple docked poses for each molecule and write out all energy terms to a SD file, type:

```
% VinaPerformDocking.py -g SampleACE2RefLigand.pdb
  -r SampleACE2Receptor.pdbqt  --numPoses 5 --energyComponents yes
  -i SampleACE2Ligands.sdf -o SampleACE2LigandsOut.sdf
```

To run the first example for docking molecules using Vinardo forcefield and write out a SD file, type:

```
% VinaPerformDocking.py -f Vinardo -g SampleACE2RefLigand.pdb
  -r SampleACE2Receptor.pdbqt -i SampleACE2Ligands.sdf
  -o SampleACE2LigandsOut.sdf
```

To run the first example for docking molecules using AD4 forcefield relying on the presence of affinity maps in the working directory and write out a SD file, type:

```
% VinaPerformDocking.py -f AD4 -g SampleACE2RefLigand.pdb
  -r SampleACE2Receptor -i SampleACE2Ligands.sdf
  -o SampleACE2LigandsOut.sdf
```

To run the first example for docking molecules using a set of explicit values for grid dimensions and write out a SD file, type:

```
% VinaPerformDocking.py -g "41.399, 5.851, 28.082"
  --gridSize "25.0, 25.0, 25.0" --gridSpacing 0.375
  -r SampleACE2Receptor.pdbqt -i SampleACE2Ligands.sdf
  -o SampleACE2LigandsOut.sdf
```

To run the first example for only scoring molecules already positioned in the binding pocket and write out a SD file, type:

```
% VinaPerformDocking.py -m ScoreOnly -g SampleACE2RefLigand.sdf
  -r SampleACE2Receptor.pdbqt -i SampleACE2RefLigandWithHs.sdf
  -o SampleACE2LigandsOut.sdf
```

To run the first example for docking molecules to increase the accuracy of pose predictions and write out a SD file, type:

```
% VinaPerformDocking.py -g SampleACE2RefLigand.pdb
  -r SampleACE2Receptor.pdbqt --exhaustiveness 24
  -i SampleACE2Ligands.sdf -o SampleACE2LigandsOut.sdf
```

To run the first example in multiprocessing mode on all available CPUs without loading all data into memory, a single thread for Vina docking, and write out a SD file, type:

```
% VinaPerformDocking.py -g SampleACE2RefLigand.pdb
  -r SampleACE2Receptor.pdbqt --mp yes
  -i SampleACE2Ligands.sdf -o SampleACE2LigandsOut.sdf
```

To run the first example in multiprocessing mode on all available CPUs by loading all data into memory, a single thread for Vina, and write out a SD file, type:

```
% VinaPerformDocking.py -g SampleACE2RefLigand.pdb
  -r SampleACE2Receptor.pdbqt --mp yes --mpParams "inputDataMode,
  InMemory" -i SampleACE2Ligands.sdf -o SampleACE2LigandsOut.sdf
```

To run the first example in multiprocessing mode on specific number of CPUs and chunk size without loading all data into memory along with a specific number of threads for Vina docking and write out a SD file, type:

```
% VinaPerformDocking.py -g SampleACE2RefLigand.pdb
  -r SampleACE2Receptor.pdbqt --mp yes --mpParams "inputDataMode,lazy,
  numProcesses,4,chunkSize,2" --numThreads 2
  -i SampleACE2Ligands.sdf -o SampleACE2LigandsOut.sdf
```

To run the first example for docking molecules employing a flexible portion of the receptor corresponding to ARG273 and write out a SD file, type:

```
% VinaPerformDocking.py -g SampleACE2RefLigand.pdb
  -r SampleACE2RigidReceptor.pdbqt
  --receptorFlexFile SampleACE2FlexReceptor.pdbqt
  -i SampleACE2Ligands.sdf -o SampleACE2LigandsOut.sdf
```

To run the first example for docking molecules using specified parameters and write out a SD file, type:

```
% VinaPerformDocking.py -g "41.399, 5.851, 28.082"
  --gridSize "25.0, 25.0, 25.0" --gridSpacing 0.375 --energyComponents
  yes  --exhaustiveness 32 --forcefield Vina --mode dock --numPoses 2
  --numThreads 4 --randomSeed 42 --validateMolecules no
  --vinaVerbosity  0 -r SampleACE2Receptor.pdbqt
  -i SampleACE2Ligands.sdf -o SampleACE2LigandsOut.sdf
```

AUTHOR

Manish Sud(msud@san.rr.com)

ACKNOWLEDGMENTS

Diogo Santos-Martins and Stefano Forli

SEE ALSO

PyMOLConvertLigandFileFormat.py, PyMOLExtractSelection.py, PyMOLInfoMacromolecules.py, PyMOLVisualizeMacromolecules.py, RDKitConvertFileFormat.py, RDKitEnumerateTautomers.py, RDKitGenerateConformers.py, RDKitPerformMinimization.py, RDKitPerformConstrainedMinimization.py

COPYRIGHT

Copyright (C) 2025 Manish Sud. All rights reserved.

The functionality available in this script is implemented using AutoDockVina and Meeko, open source software packages for docking, and RDKit, an open source toolkit for cheminformatics developed by Greg Landrum.

This file is part of MayaChemTools.

MayaChemTools is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.