NAME

RDKitGenerateConstrainedConformers.py - Generate constrained molecular conformations

SYNOPSIS

RDKitGenerateConstrainedConformers.py [--addHydrogens <yes or no>] [--conformerGenerator <SDG, KDG, ETDG, ETKDG, ETKDGv2>] [--forceField <UFF, or MMFF>] [--forceFieldMMFFVariant <MMFF94 or MMFF94s>] [--energyOut <yes or no>] [--enforceChirality <yes or no>] [--energyRMSDCutoff <number>] [--energyWindow <number>] [--infileParams <Name,Value,...>] [--maxConfs <number>] [--mcsParams <Name,Value,...>] [--mp <yes or no>] [--mpParams <Name,Value,...>] [--outfileParams <Name,Value,...>] [--overwrite] [--quiet <yes or no>] [--removeHydrogens <yes or no>] [--scaffold <auto or SMARTS>] [--scaffoldRMSDOut <yes or no>] [--useTethers <yes or no>] [-w <dir>] -i <infile> -r <reffile> -o <outfile>

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

DESCRIPTION

Generate molecular conformations by performing a constrained energy minimization against a reference molecule. An initial set of 3D conformers are generated for the input molecules using distance geometry. A common core scaffold, corresponding to a Maximum Common Substructure (MCS) or an explicit SMARTS pattern, is identified between a pair of input and reference molecules. The core scaffold atoms in input molecules are aligned against the same atoms in the reference molecule. The energy of aligned structures are minimized using the forcefield to generate the final 3D structures.

The supported input file formats are: Mol (.mol), SD (.sdf, .sd), SMILES (.smi, .csv, .tsv, .txt)

The supported output file formats are: SD (.sdf, .sd)

OPTIONS

-a, --addHydrogens <yes or no> [default: yes]

Add hydrogens before minimization.

-c, --conformerGenerator <text> [default: ETKDGv2]

Conformation generation methodology for generating initial 3D coordinates for molecules in input file. A common core scaffold is identified between a a pair of input and reference molecules. The atoms in common core scaffold of input molecules are aligned against the reference molecule followed by energy minimization to generate final 3D structure.

The possible values along with a brief description are shown below:

```
SDG: Standard Distance Geometry
KDG: basic Knowledge-terms with Distance Geometry
ETDG: Experimental Torsion-angle preference with Distance Geometry
ETKDG: Experimental Torsion-angle preference along with basic
    Knowledge-terms and Distance Geometry [Ref 129]
ETKDGv2: Experimental Torsion-angle preference along with basic
    Knowledge-terms and Distance Geometry [Ref 167]
```

-f, --forceField <UFF, MMFF> [default: MMFF]

Forcefield method to use for constrained energy minimization. Possible values: Universal Force Field (UFF) [Ref 81] or Merck Molecular Mechanics Force Field [Ref 83-87] .

--forceFieldMMFFVariant <MMFF94 or MMFF94s> [default: MMFF94]

Variant of MMFF forcefield to use for energy minimization.

--energyOut <yes or no> [default: No]

Write out energy values.

--enforceChirality <yes or no> [default: Yes]

Enforce chirality for defined chiral centers.

--energyRMSDCutoff <number> [default: 0.5]

RMSD cutoff for retaining conformations after embedding and energy minimization. Possible values: A number or None

The default is to keep only those conformations which are different from the lowest energy conformation by the specified RMSD cutoff. The None value may be used to keep all minimized conformations with in the specified energy window from the lowest energy conformation. The lowest

energy conformation is always retained.

--energyWindow <number> [default: 20]

Energy window in kcal/mol for selecting conformers.

-e, --examples

Print examples.

-h, --help

Print this help message.

-i, --infile <infile>

Input file name.

--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,yes,sanitize,yes,strictParsing,yes

SMILES: smilesColumn,1,smilesNameColumn,2,smilesDelimiter,space,
    smilesTitleLine,auto,sanitize,yes
```

Possible values for smilesDelimiter: space, comma or tab.

--maxConfs <number> [default: 50]

Maximum number of conformations to generate for each molecule by conformation generation methodology for initial 3D coordinates. A constrained minimization is performed using the specified forcefield and the lowest energy conformation is written to the output file.

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

Parameter values to use for identifying a maximum common substructure (MCS) in between a pair of reference and input molecules.In general, it is a comma delimited list of parameter name and value pairs. The supported parameter names along with their default values are shown below:

```
atomCompare,CompareElements,bondCompare,CompareOrder,
maximizeBonds,yes,matchValences,yes,matchChiralTag,no,
minNumAtoms,1,minNumBonds,0,ringMatchesRingOnly,yes,
completeRingsOnly,yes,threshold,1.0,timeOut,3600,seedSMARTS,none
```

Possible values for atomCompare: CompareAny, CompareElements, CompareIsotopes. Possible values for bondCompare: CompareAny, CompareOrder, CompareOrderExact.

A brief description of MCS parameters taken from RDKit documentation is as follows:

```
atomCompare - Controls match between two atoms
bondCompare - Controls match between two bonds
maximizeBonds - Maximize number of bonds instead of atoms
matchValences - Include atom valences in the MCS match
matchChiralTag - Include atom chirality in the MCS match
minNumAtoms - Minimum number of atoms in the MCS match
minNumBonds - Minimum number of bonds in the MCS match
ringMatchesRingOnly - Ring bonds only match other ring bonds
completeRingsOnly - Partial rings not allowed during the match
threshold - Fraction of the dataset that must contain the MCS
seedSMARTS - SMARTS string as the seed of the MCS
timeout - Timeout for the MCS calculation in seconds
```

--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

--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.

-o, --outfile <outfile>

Output file name.

--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.

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

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

-r, --reffile <reffile>

Reference input file name containing a 3D reference molecule. A common core scaffold must be present in a pair of an input and reference molecules. Otherwise, no constrained minimization is performed on the input molecule.

--removeHydrogens <yes or no> [default: Yes]

Remove hydrogens after minimization.

-s, --scaffold <auto or SMARTS> [default: auto]

Common core scaffold between a pair of input and reference molecules used for constrained minimization of molecules in input file. Possible values: Auto or a valid SMARTS pattern. The common core scaffold is automatically detected corresponding to the Maximum Common Substructure (MCS) between a pair of reference and input molecules. A valid SMARTS pattern may be optionally specified for the common core scaffold.

--scaffoldRMSDOut <yes or no> [default: No]

Write out RMSD value for common core alignment between a pair of input and reference molecules.

-u, --useTethers <yes or no> [default: yes]

Use tethers to optimize the final conformation by applying a series of extra forces to align matching atoms to the positions of the core atoms. Otherwise, use simple distance constraints during the optimization.

-w, --workingdir <dir>

Location of working directory which defaults to the current directory.

EXAMPLES

To generate conformers by performing constrained energy minimization for molecules in a SMILES file against a reference 3D molecule in a SD file using a common core scaffold between pairs of input and reference molecules identified using MCS, generating up to 50 conformations using ETKDG methodology followed by MMFF forcefield minimization within energy window of 20 kcal/mol and RMSD of greater than 0.5 from the lowest energy conformation, and write out a SD file:

```
% RDKitGenerateConstrainedConformers.py  -i SampleSeriesD3R.smi
  -r SampleSeriesRef3D.sdf  -o SampleOut.sdf
```

To rerun the first example in a quiet mode and write out a SD file, type:

```
% RDKitGenerateConstrainedConformers.py  -q yes -i SampleSeriesD3R.smi
  -r SampleSeriesRef3D.sdf  -o SampleOut.sdf
```

To rerun the first example in multiprocessing mode on all available CPUs without loading all data into memory and write out a SD file, type:

```
% RDKitGenerateConstrainedConformers.py  --mp yes -i SampleSeriesD3R.smi
  -r SampleSeriesRef3D.sdf  -o SampleOut.sdf
```

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

```
% RDKitGenerateConstrainedConformers.py  --mp yes --mpParams
  "inputDataMode,InMemory" -i SampleSeriesD3R.smi
  -r SampleSeriesRef3D.sdf  -o SampleOut.sdf
```

To rerun the first example in multiprocessing mode on specific number of CPUs and chunk size without loading all data into memory and write out a SD file, type:

```
% RDKitGenerateConstrainedConformers.py  --mp yes --mpParams
  "inputDataMode,Lazy,numProcesses,4,chunkSize,8"
  -i SampleSeriesD3R.smi -r SampleSeriesRef3D.sdf  -o SampleOut.sdf
```

To rerun the first example using an explicit SMARTS string for a common core scaffold and write out a SD file, type:

```
% RDKitGenerateConstrainedConformers.py  --scaffold
  "c2cc(-c3nc(N)ncc3)cn2" -i SampleSeriesD3R.smi
  -r SampleSeriesRef3D.sdf  -o SampleOut.sdf
```

To rerun the first example using molecules in a CSV SMILES file, SMILES strings in column 1, name in column2, and write out a SD file, type:

```
% RDKitGenerateConstrainedConformers.py  --infileParams
  "smilesDelimiter,comma,smilesTitleLine,yes,smilesColumn,1,
  smilesNameColumn,2" -i SampleSeriesD3R.csv -r SampleSeriesRef3D.sdf
  -o SampleOut.sdf
```

To generate constrained conformers for molecules in a SD file against a reference 3D molecule in a SD file using a common core scaffold between pairs of input and reference molecules identified using MCS, generating up to 10 conformations using SDG methodology followed by UFF forcefield minimization, conformations with in an energy window of 10 kcal/mol and RMSD of greater that 1, and write out a SD file containing minimum energy structure along with energy and embed RMS values corresponding to each constrained molecule, type:

```
% RDKitGenerateConstrainedConformers.py  --maxConfs 10  -c SDG -f UFF
  --scaffoldRMSDOut yes --energyOut yes --energyRMSDCutoff 1.0
  --energyWindow 10 -i SampleSeriesD3R.sdf -r SampleSeriesRef3D.sdf
 -o SampleOut.sdf
```

AUTHOR

Manish Sud(msud@san.rr.com)

SEE ALSO

RDKitCalculateRMSD.py, RDKitCalculateMolecularDescriptors.py, RDKitCompareMoleculeShapes.py, RDKitConvertFileFormat.py, RDKitGenerateConformers.py, RDKitPerformConstrainedMinimization.py

COPYRIGHT

Copyright (C) 2026 Manish Sud. All rights reserved.

The functionality available in this script is implemented using 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.