GOLD Configuration File User Guide

- param_file
- set_protein_atom_types
- set_ligand_atom_types
- directory
- tordist_file
- make_subdirs
- save_lone_pairs
- bestranking_list_filename
- fit_points_file
- read_fitpts

Flags

- internal_ligand_h_bonds
- flip_free_corners
- match_ring_templates
- flip_amide_bonds
- flip_planar_n
- flip_pyramidal_n
- rotate_carboxylic_oh
- use_tordist
- fix_rotatable bond
- postprocess_bonds
- rotatable_bond_override_file
- diverse_solutions
- divsol_cluster_size
- divsol_rmsd
- solvate_all
- fix_all_protein_rotatable_bonds

Termination

- early_termination
- n_top_solutions
- rms_tolerance

Constraints

- constraint distance
- constraint h_bond
- constraint protein_h_bond
- constraint sphere
- constraint pharmacophore
- constraint substructure
- constraint similarity
- constraint scaffold

- interaction_restraint_weight
- constraint interaction_restraint
- force_constraints

Covalent Bonding

- covalent
- covalent_protein_atom_no
- covalent_ligand_atom_no
- covalent_substructure_filename
- covalent_substructure_atom_no
- covalent_substructure
- covalent_topology

Save Options

- save_score_in_file
- save_protein_torsions
- concatenated_output
- clean_up_option delete_all_solutions
- clean_up_option save_fitness_better_than
- clean_up_option save_top_n_solutions
- clean_up_option save_best_ligands
- clean_up_option delete_redundant_log_files
- clean_up_option save_clustered_solutions
- clean_up_option delete_empty_directories
- clean_up_option delete_rank_file
- clean_up_option delete_symlinks
- clean_up_option delete_all_log_files
- clean_up_option delete_all_initialised_ligands
- output_file_format
- per_atom_scores
- save_per_atom_scores_to_charge_field

Write Options

- write_options

Fitness Function Settings

- initial_virtual_pt_match_max
- relative_ligand_energy
- gold_fitfunc_path
- docking_fitfunc_path
- docking_param_file
- rescore_fitfunc_path
- rescore_param_file
- score_param_file

GOLD Configuration File User Guide

A Component of the CSD-Discovery Suite

2022.3 CSD Release

Registered Charity No 800579

Conditions of Use

The GOLD program, Hermes visualiser, associated documentation and software, are copyright works of CCDC Software Limited and its licensors and all rights are protected. Use of the Program is permitted solely in accordance with a valid Software Licence Agreement or a valid Licence and Support Agreement with CCDC Software Limited or a valid Licence of Access to the CSD Portfolio with CCDC and the Program is proprietary. All persons accessing the Program should make themselves aware of the conditions contained in the Software Licence Agreement or Licence and Support Agreement or Licence of Access Agreement.

In particular:

- The Program is to be treated as confidential and may NOT be disclosed or re-distributed in any form, in whole or in part, to any third party.
- No representations, warranties, or liabilities are expressed or implied in the supply of the Program by CCDC Software Ltd., its servants or agents, except where such exclusion or limitation is prohibited, void or unenforceable under governing law.
- GOLD © 2022 CCDC Software Ltd.
- Hermes © 2022 CCDC Software Ltd.

Implementation of ChemScore, Heme, Kinase and Astex Statistical Potential scoring functions and the Diverse Solutions code within GOLD © 2001-2022 Astex Therapeutics Ltd.

Licences may be obtained from:

CCDC Software Ltd.

12 Union Road

Cambridge CB2 1EZ

United Kingdom

Web: www.ccdc.cam.ac.uk

Telephone: +44-1223-336408

Email: admin@ccdc.cam.ac.uk

Overview: Using Configuration Files with GOLD

The configuration file is a text file which specifies the GOLD calculation that is to be run, including details of the ligand, the protein binding site, the fitness-function parameter file to be used, the torsion distribution file to be used, and the genetic algorithm parameters. Although the file can be generated with a standard text editor, the easiest way to create it is to use the GOLD graphical user interface.

Any settings that have been defined in the GOLD interface can be saved as a configuration file by selecting the **Save** button located next to the **Conf file** entry box at the top of the **GOLD Setup** window. Alternatively, you will be prompted to save the file if you start a GOLD job from the interface by selecting either **Run GOLD**, or **Run GOLD in the background**.

By default, the configuration file will be saved in the directory from which GOLD was opened and will be called `gold.conf`. Use the **Conf file** entry box at the top of the **GOLD Setup** window to change the file name and/or directory (any file name can be used).

Certain advanced functionality in GOLD is only available by directly editing the GOLD configuration file, i.e. some functionality is not exposed in the GOLD graphical user interface. To use these features you will therefore need to set up the GOLD job as normal in the graphical interface, save the configuration file, then manually edit this file.

Once a configuration file has been created, it can be re-used, either as a quick way of reading program settings into the GOLD front end or to run GOLD from the command line.

Running GOLD Using a Configuration File

To load a previously created configuration file into GOLD interface, enter the file name into the **Conf file** entry box at the top of the **GOLD Setup** window. Alternatively, click on the **Load** button and use the file selection window to choose the file. The parameters read in from the configuration file will overwrite any parameters that have already been set in the GOLD front end.

To run GOLD from the command line using a configuration file, issue the following command:

- Unix Platforms:

  GOLD can be run directly in the background by using a simple command available in `$GOLD_DIR/bin`:

  ```
  gold_auto gold.conf &
  ```

  where `gold.conf` is the name of a configuration file and `$GOLD_DIR` = `<Installation folder>/Discovery_2022/`. The `<Installation folder>` is most likely to be: `/home/username/CCDC/`

- Windows Platforms:

  GOLD can be run on Windows by starting a command prompt, navigating to the directory containing the `gold.conf` file and running the following command (all on one line):

  ```
  "C:\Program
  Files\CCDC\Discovery_2022\GOLD\gold\d_win32\bin\gold_win32.exe"
  ```

The above command assumes that GOLD is installed in the default installation directory and that the configuration file is called `gold.conf`. If another name has been used for the `gold.conf`, (e.g. `new_conf_filename.conf`), this will have to be specified:

```
"C:\Program
  Files\CCDC\Discovery_2022\GOLD\gold\d_win32\bin\gold_win32.exe"
  new_conf_filename.conf
```

- macOS:

  Please set `$GOLD_DIR` to e.g. `<Installation folder>/Discovery_2022/GOLD` where `<Installation folder>` is most likely to be : `/Applications/CCDC/`

  Please make sure `$CSDHOME` is set to e.g. `<Installation folder>/`

  You can then run the `gold_auto` script, located in `$GOLD_DIR/bin`:

  ```
  gold_auto gold.conf &
  ```

Format of the GOLD Configuration File

The GOLD configuration file is a plain text file containing instructions with one instruction per line. Instructions are case sensitive.

Typically, the order in which instructions are provided is not important.

Any text that is not recognised will cause a warning to be issued:

```
Warning message:
can't process configuration line:
details
```

Any text appearing after a # will be treated as a comment and ignored.

Automatic Settings

autoscale

```
autoscale = <value>
default: 0 (off)
```

Setting all population and genetic operators to `auto` will instruct GOLD to automatically calculate the optimal number of operations for each ligand, thereby making the most efficient use of search time. When using automatic settings the search efficiency (`autoscale`) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results.

The value of autoscale can be set between 0.01 and 5.0. When set at 1.0 (search efficiency = 100%) GOLD will attempt to apply optimal settings for each ligand. For a ligand with five rotatable bonds this will be around 30,000 GA operations. Note: the exact number of GA operations contributed, e.g. for each rotatable bond in the ligand, are defined in the `gold.params` file. If the Search efficiency were set to 0.5, then GOLD will perform around 15,000 operations thereby speeding up the docking by a factor of two (however the search space will be less well explored). Similarly, by setting a Search efficiency greater than 1.0, it is possible to make the search more exhaustive (but slower).

When using `autoscale` it is further possible to ensure that every ligand is subjected to a user-specified minimum and/or maximum number of operations (see `autoscale_nops_min`) and (see `autoscale_nops_max`).

Related instructions:

- `autoscale_nops_min`
- `autoscale_nops_max`
- `popsiz`
- `select_pressure`

- n_islands
- match_ring_templates
- niche_siz
- pt_crosswt
- allele_mutatewt
- migratewt

autoscale_nops_max

```
autoscale_nops_max = <value>
default: 0 (off)
```

When using automatic (ligand dependent) GA parameter settings the search efficiency (see autoscale) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results. When using `autoscale` the maximum number of GA operations performed during the docking run will be updated automatically according to the `autoscale` value that is set. The automatic preset can be overridden to ensure that every ligand is subjected to no more than `autoscale_nops_max` operations. Similarly, the minimum number of operations can be specified (see autoscale_nops_min).

Related Instructions:

- autoscale
- autoscale_nops_min
- popsiz
- select_pressure
- n_islands
- match_ring_templates
- niche_siz

- pt_crosswt
- allele_mutatewt
- migratewt

autoscale_nops_min

```
autoscale_nops_min = <value>
default: 0 (off)
```

When using automatic (ligand dependent) GA parameter settings the search efficiency (see autoscale) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results. When using autoscale the minimum number of GA operations performed during the docking run will be updated automatically according to the autoscale value that is set. The automatic preset can be overridden to ensure that every ligand is subjected to at least autoscale_nops_min operations. Similarly, the maximum number of operations can be specified (see autoscale_nops_max).

Related Instructions:

- autoscale
- autoscale_nops_max
- popsiz
- select_pressure
- n_islands
- match_ring_templates
- niche_siz
- pt_crosswt
- allele_mutatewt
- migratewt

Population

popsiz

```
popsiz = <value> | auto
default: 100
```

The genetic algorithm maintains a set of possible solutions to the problem. Each possible solution is known as a chromosome and the set of solutions is termed a population. The value of `popsiz` (population size) is the number of chromosomes in the population. If the number of islands (see `n_islands`) is greater than one (i.e. the genetic algorithm is split over two or more islands), then `popsiz` is the population on each island.

Optimum values of the genetic algorithm parameters are highly correlated, you are therefore recommended to use automatic (ligand dependent) GA parameter settings (see below) or one of the default parameter sets offered via the front end.

Setting all population and genetic operators to `auto` will instruct GOLD to automatically calculate the optimal number of operations for each ligand, thereby making the most efficient use of search time. When using automatic settings the search efficiency (see `autoscale`) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results.

Related Instructions:

- `select_pressure`
- `n_islands`
- `match_ring_templates`
- `niche_siz`
- `pt_crosswt`
- `allele_mutatewt`
- `migratewt`

- autoscale

select_pressure

```
select_pressure = <value> | auto
default: 1.1
```

Each of the genetic operations (crossover, migration, mutation) takes information from parent chromosomes and assembles this information in child chromosomes. The child chromosomes then replace the worst members of the population. The selection of parent chromosomes is biased towards those of high fitness, i.e. a fit chromosome is more likely to be a parent than an unfit one. The selection pressure is defined as the ratio between the probability that the most fit member of the population is selected as a parent to the probability that an average member is selected as a parent.

Optimum values of the genetic algorithm parameters are highly correlated, you are therefore recommended to use automatic (ligand dependent) GA parameter settings (see below) or one of the default parameter sets offered via the front end.

Setting all population and genetic operators to `auto` will instruct GOLD to automatically calculate the optimal number of operations for each ligand, thereby making the most efficient use of search time. When using automatic settings the search efficiency (see autoscale) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results.

Related Instructions:

- popsiz
- n_islands
- match_ring_templates
- niche_siz
- pt_crosswt
- allele_mutatewt

- migratewt
- autoscale

n_islands

```
n_islands = <value> | auto
default: 5
```

Rather than maintaining a single population, the genetic algorithm can maintain a number of populations that are arranged as a ring of islands. Specifically, the algorithm maintains `n_islands` populations, each of size `popsiz` (see popsiz). Individuals can migrate between adjacent islands using the migration operator (see migratewt).

Optimum values of the genetic algorithm parameters are highly correlated, you are therefore recommended to use automatic (ligand dependent) GA parameter settings (see below) or one of the default parameter sets offered via the front end.

Setting all population and genetic operators to `auto` will instruct GOLD to automatically calculate the optimal number of operations for each ligand, thereby making the most efficient use of search time. When using automatic settings the search efficiency (see autoscale) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results.

Related Instructions:

- popsiz
- select_pressure
- match_ring_templates
- niche_siz
- pt_crosswt
- allele_mutatewt
- migratewt

- autoscale

maxops

```
maxops = <value> | auto
default: 100000
```

The genetic algorithm starts off with a random population (each value in every chromosome is set to a random number). Genetic operations (crossover, migration, mutation) are then applied iteratively to the population. `maxops` is the number of operators that are applied over the course of a GA run. It is the key parameter in determining how long a GOLD run will take.

Optimum values of the genetic algorithm parameters are highly correlated, you are therefore recommended to use automatic (ligand dependent) GA parameter settings (see below) or one of the default parameter sets offered via the front end.

Setting all population and genetic operators to `auto` will instruct GOLD to automatically calculate the optimal number of operations for each ligand, thereby making the most efficient use of search time. When using automatic settings the search efficiency (see `autoscale`) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results.

Related Instructions:

- popsiz
- select_pressure
- n_islands
- niche_siz
- pt_crosswt
- allele_mutatewt
- migratewt
- autoscale

niche_siz

```
niche_siz = <value> | auto
default: 2
```

Niching is a common technique used in genetic algorithms to preserve diversity within the population. In GOLD, two individuals share the same niche if the rmsd between the coordinates of their donor and acceptor atoms is less than 1.0 Å. When adding a new individual to the population, a count is made of the number of individuals in the population that inhabit the same niche as the new chromosome. If there are more than `niche_siz` individuals in the niche, then the new individual replaces the worst member of the niche rather than the worst member of the total population.

Optimum values of the genetic algorithm parameters are highly correlated, you are therefore recommended to use automatic (ligand dependent) GA parameter settings (see below) or one of the default parameter sets offered via the front end.

Setting all population and genetic operators to `auto` will instruct GOLD to automatically calculate the optimal number of operations for each ligand, thereby making the most efficient use of search time. When using automatic settings the search efficiency (see `autoscale`) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results.

Related Instructions:

- `popsiz`
- `select_pressure`
- `n_islands`
- `match_ring_templates`
- `pt_crosswt`
- `allele_mutatewt`
- `migratewt`
- `autoscale`

Genetic Operators

pt_crosswt

```
pt_crosswt = <value> | auto
default: 95
```

The operator weights for the parameters mutate, migrate and crossover govern the relative frequencies of the three types of operations that can occur during a genetic optimisation: point mutation of the chromosome, migration of a population member from one island to another, and crossover (sexual mating) of two chromosomes. Each time the genetic algorithm selects an operator, it does so at random. Any bias in this choice is determined by the operator weights. For example, if Mutate is 40 and Crossover is 10 then, on average, four mutations will be applied for every crossover.

Optimum values of the genetic algorithm parameters are highly correlated, you are therefore recommended to use automatic (ligand dependent) GA parameter settings (see below) or one of the default parameter sets offered via the front end.

Setting all population and genetic operators to `auto` will instruct GOLD to automatically calculate the optimal number of operations for each ligand, thereby making the most efficient use of search time. When using automatic settings the search efficiency (see `autoscale`) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results.

Related Instructions:

- `popsiz`
- `select_pressure`
- `n_islands`
- `match_ring_templates`
- `niche_siz`
- `allele_mutatewt`

- migratewt
- autoscale

allele_mutatewt

```
allele_mutatewt = <value> | auto
default: 95
```

The operator weights for the parameters mutate, migrate and crossover govern the relative frequencies of the three types of operations that can occur during a genetic optimisation: point mutation of the chromosome, migration of a population member from one island to another, and crossover (sexual mating) of two chromosomes. Each time the genetic algorithm selects an operator, it does so at random. Any bias in this choice is determined by the operator weights. For example, if Mutate is 40 and Crossover is 10 then, on average, four mutations will be applied for every crossover.

Optimum values of the genetic algorithm parameters are highly correlated, you are therefore recommended to use automatic (ligand dependent) GA parameter settings (see below) or one of the default parameter sets offered via the front end.

Setting all population and genetic operators to `auto` will instruct GOLD to automatically calculate the optimal number of operations for each ligand, thereby making the most efficient use of search time. When using automatic settings the search efficiency (see autoscale) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results.

Related Instructions:

- popsiz
- select_pressure
- n_islands
- match_ring_templates
- niche_siz

- pt_crosswt
- migratewt
- autoscale

migratewt

```
migratewt = <value> | auto
default: 10
```

The operator weights for the parameters mutate, migrate and crossover govern the relative frequencies of the three types of operations that can occur during a genetic optimisation: point mutation of the chromosome, migration of a population member from one island to another, and crossover (sexual mating) of two chromosomes. Each time the genetic algorithm selects an operator, it does so at random. Any bias in this choice is determined by the operator weights. For example, if Mutate is 40 and Crossover is 10 then, on average, four mutations will be applied for every crossover. The migrate weight should be zero if there is only one island, otherwise migration should occur about 5% of the time.

Optimum values of the genetic algorithm parameters are highly correlated, you are therefore recommended to use automatic (ligand dependent) GA parameter settings (see below) or one of the default parameter sets offered via the front end.

Setting all population and genetic operators to `auto` will instruct GOLD to automatically calculate the optimal number of operations for each ligand, thereby making the most efficient use of search time. When using automatic settings the search efficiency (see `autoscale`) can be used to control the speed of docking and the predictive accuracy (i.e. the reliability) of the results.

Related Instructions:

- popsiz
- select_pressure
- n_islands

- match_ring_templates
- niche_siz
- allele_mutatewt
- pt_crosswt
- autoscale

Flood Fill

radius

```
radius = <value>
default: 10
```

The binding site can be defined as all atoms within <value> Å of a specified central point (e.g. this could be a protein atom close to the centre of the active site, or a point, defined by X,Y,Z coordinates). The radius should be large enough to contain any possible binding mode of the ligand.

Related instructions:

- floodfill_atom_no
- origin

origin

```
origin = <x value> <y value> <z value>
default: 0.0 0.0 0.0
```

When used in combination with `floodfill_center = point` (see floodfill_center) this option will define the binding site from a single point. The orthogonal x,y,z coordinates of a single solvent-accessible point approximately at the centre of the active site

should be specified. The binding site will subsequently be defined as all atoms that lie within a given radius (see radius) of the specified point.

Related instructions:

- floodfill_center
- radius

do_cavity

```
do_cavity = <option>
default: 1 (on)
options: 0 (off) | 1 (on)
```

If this option is switched on a cavity detection algorithm will be used to restrict the binding site definition to concave, solvent-accessible surfaces. The algorithm LIGSITE is used for the automatic detection of potential small molecule binding sites in proteins, see M. Hendlich, F. Rippmann and G. Barnickel. J. Mol. Graph. Model., **15**, 359-63, 389 (1997).

Related instructions:

- cavity_file
- floodfill_atom_no
- origin

floodfill_atom_no

```
floodfill_atom_no = <atom id>
default: 0 (off)
```

When used in combination with `floodfill_center = atom` (see floodfill_center) this option will define the binding site from a single protein atom. The atom id (as it appears in the protein input file) of a single solvent-accessible atom close to the centre of the

active site of the protein should be specified. The binding site will subsequently be defined as all atoms that lie within a given radius (see `radius`) of the specified protein atom.

When used in combination with `floodfill_center = residue` (see `floodfill_center`) this option will define the binding site from a single residue. The atom id (as it appears in the protein input file) of any atom within the residue from which you want to define the active site should be specified. All protein atoms that lie within a given radius (see `radius`) of each atom in the selected residue are found, these atoms plus the atoms of their associated residues are then used for the active site definition.

Related instructions:

- `do_cavity`
- `floodfill_center`
- `radius`

cavity_file

`cavity_file = <filename>`

`cavity_file` is used to provide GOLD with a binding site definition and must be followed by a filename, e.g. `cavity_file = Z:\GOLD\datafiles\ligand_reference.mol2`

When used in combination with `floodfill_center = cavity_from_ligand` (see `floodfill_center`) this option will define the binding site from a specified reference ligand. This could be a ligand in a known binding mode, or the co-crystallised ligand, or a cofactor.

By default, only protein atoms that lie within 6.0 Å of any ligand atom are included. To instead include protein atoms within a user-specified distance of the ligand, use the following:

`floodfill_center = cavity_from_ligand <distance> atoms`

where `<distance>` is the distance threshold and the (optional) keyword `atoms` instructs GOLD not to include all atoms of each residue found (see below).

To use protein atoms within a user-specified cavity distance threshold of each ligand atom plus all atoms of their associated residues, use the keyword `resid`, as in the following:

```
floodfill_center = cavity_from_ligand <distance> resid
```

When used in combination with `floodfill_center = file` (see `floodfill_center`) this option will define the binding site from a list of protein atoms. The file specified should contain the atom id (as it appears in the protein input file) of every solvent-accessible atom which is required to explicitly define the protein active site (all acceptor and donor hydrogen atoms available to the ligand are taken from the file). Multiple atom numbers are permitted on each line in the file.

When used in combination with `floodfill_center = list_of_residues` (see `floodfill_center`) this option will define the binding site from a specified list of residues. The residues can be extracted from any text file, including a standard GOLD solution file (GOLD writes the active site residues list to the solution files if output of rotatable hydrogens is turned on). The following formatting restrictions apply:

The list must begin with the following tag on its own line:

```
> <Gold.Protein.ActiveResidues>
```

The list must end with a blank line (or the end of the text file). GOLD will read multiple residue names from one line, but lines must not exceed 250 characters in length. Residue names must be separated by a space, for example:

```
> <Gold.Protein.ActiveResidues>
HIS69 ARG71 GLU72 ARG127 ASN144 ARG145 GLY155 ALA156 GLU163 THR164
HIS196 SER197 TYR198 SER199 LEU201 LEU203 ILE243 ILE244 ILE247
        TYR248
GLN249 ALA250 GLY253 SER254 ILE255 THR268 GLU270 PHE279 ZN309
```

The list should contain all the residues which are required to explicitly define the protein active site (all acceptor and donor hydrogen atoms available to the ligand are taken from the list).

Related instructions:

- do_cavity
- floodfill_center

floodfill_center

```
floodfill_center = < atom | cavity_from_ligand | file |
list_of_residues | point | residue >
```

```
default: point
```

floodfill_center determines the method used to define the binding site. The possible settings are:

- atom: used to define the binding site from a single protein atom (see floodfill_atom_no)
- cavity_from_ligand: used to define the binding site from a reference ligand (see cavity_file)
- file: used to define the binding site from a list of protein atoms (see cavity_file)
- list_of_residues: used to define the binding site from a list of residues (see cavity_file)
- point: used to define the binding site from a single point using orthogonal x,y,z coordinates (see origin)
- residue: used to define the binding site from a specified residue (see floodfill_atom_no)

Related instructions:

- cavity_file
- floodfill_atom_no

- origin

Data Files

ligand_data_file

`ligand_data_file <filename> <no. of GA runs> [start_at_ligand <value> finish_at_ligand <value>]`

`ligand_data_file` is used to provide GOLD with the ligand file(s) to be docked and must be followed by a filename. The value of `<no. of GA runs>` is the number of times each ligand is to be docked and must also be specified, e.g.

```
ligand_data_file Z:\GOLD\datafiles\ligand.mol2 10
```

Acceptable ligand file formats are MOL2 (i.e. Tripos format), or MOL (i.e. MDL SD format). Before being used with GOLD the ligand input file(s) must have been prepared in accordance with the guidelines provided, using a good modelling package.

Any number of ligands can be docked, either by specifying a directory containing several ligand files, by specifying a single file containing several ligands (i.e. a multi-MOL2 or SD file), or by specifying several individual files. When specifying several individual files multiple lines must be used e.g.

```
ligand_data_file Z:\GOLD\datafiles\ligand1.mol2 10
ligand_data_file Z:\GOLD\myfiles\ligand2.mol2 20
```

Note: By specifying multiple individual files (on separate lines) it is possible to have different values of `<no. of GA runs>` for individual ligands.

When using a single file containing several ligands (i.e. a multi-MOL2 or SD file) it is possible to only dock specific ligands in that file. The ligand you wish to start and finish docking at can be specified using the following option:

`[start_at_ligand <value> finish_at_ligand <value>]`

Where `<value>` is the number relating to the position of the ligand within the file. Unless specified otherwise GOLD will, by default, start at the first ligand and finish at the last ligand in the file. Therefore, it is possible to omit the `finish_at_ligand` keyword, in which case GOLD will finish at the last ligand in the file.

ligand_reference_file

```
ligand_reference_file = <filename>
```

`ligand_reference_file` is used to provide GOLD with a file containing a reference ligand (e.g. a crystallographically observed ligand pose). ligand_reference_file must be followed by a filename, e.g.:

```
ligand_reference_file = Z:\GOLD\datafiles\cryst_observed_pose.mol2
```

The ligand reference file will be used to perform automated RMSd calculations against GOLD solution(s). For each GOLD solution the resultant RMSd with respect to the reference ligand will be written to the files containing the fitness function rankings, i.e. the ligand rank file (`.rnk`) and `bestranking.1st` file.

param_file

```
param_file = <filename> | DEFAULT
default: DEFAULT
```

`param_file` is used to provide GOLD with the location of the parameter file. The parameter file contains all of the parameters used by GOLD (e.g. hydrogen bond energies, atom radii and polarisabilities, torsion potentials, hydrogen bond directionalities, etc.) It also contains parameters that control the general behaviour of GOLD, e.g. whether the final solution from a genetic algorithm run is to be minimised via a Simplex procedure before being saved.

If param_file is set to `DEFAULT` then the standard parameter file `gold.params` (supplied in the GOLD distribution) is used. The parameter file can be customised by copying it, editing the copy, and instructing GOLD to use the edited file, e.g.

```
param_file = Z:\GOLD\datafiles\custom.params
```

Note: Editing the parameters file may cause GOLD to behave in unexpected ways. No guarantee can be given that the program will behave reliably with anything other than the default parameterisation. For more information see the comments in the parameter file itself, `gold.params`.

set_protein_atom_types

```
set_protein_atom_types = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

Each protein atom must be assigned an atom type which is used, for example, to determine whether the atom is capable of forming hydrogen bonds (GOLD atom typing is based on SYBYL atom types). Setting `set_protein_atom_types = 1` will instruct GOLD to set atom types automatically. The atom types will be assigned from the information about element types and bond orders in the protein input file, so it is important that these are correct. When using automatic atom type assignment you still need to input the protein structure correctly, e.g. with correct bond orders and appropriate protonation states. Structure input files should be prepared in accordance with the guidelines provided using a good modelling package.

set_ligand_atom_types

```
set_ligand_atom_types = <option>
default: 1 (on)
options: 0 (off) | 1 (on)
```

Each ligand atom must be assigned an atom type which is used, for example, to determine whether the atom is capable of forming hydrogen bonds (GOLD atom typing is based on SYBYL atom types). Setting `set_ligand_atom_types = 1` will instruct GOLD to set atom types automatically. The atom types will be assigned from the information about element types and bond orders in the ligand input file(s), so it is important that these are correct. When using automatic atom type assignment you still need to input the ligand

structure(s) correctly, e.g. with correct bond orders and appropriate protonation states. Structure input files should be prepared in accordance with the guidelines provided using a good modelling package.

directory

```
directory = <filename> | .
```

`directory` is used to specify the directory to which output files will be written, e.g.

```
directory = Z:\GOLD\datafiles\output
```

Unless specified otherwise output will be written to the current working directory.

Related Instructions:

- `make_subdirs`

tordist_file

```
tordist_file = <filename> | DEFAULT
default: DEFAULT
```

Torsion angle distributions extracted from the Cambridge Structural Database (CSD) can be input to GOLD. These distributions can be used (see `use_tordist`) to restrict the ligand conformational space sampled by the genetic algorithm to those torsion angle ranges commonly observed in crystal structures.

`tordist_file` is used to provide GOLD with the location of the torsion angle distribution file. Two torsion angle distribution files are provided:

- `gold.tordist`: this is the standard torsion file and is used automatically if `tordist_file` is set to `DEFAULT`
- `mimumba.tordist`: this contains all the torsional distributions used in the MIMUMBA program (Klebe and Mietzner, J.Comput.-Aided Mol.Des., **8**, 583-606, 1994).

The torsion angle distribution file can be customised by copying it, editing the copy, and instructing GOLD to use the edited file, e.g.

```
tordist_file = Z:\GOLD\datafiles\custom.tordist
```

Note: The format of entries in the torsion angle distribution file is strict: incorrect editing of the file may cause GOLD to behave in unexpected ways.

Related Instructions:

- use_tordist

make_subdirs

```
make_subdirs = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

When more than one ligand is being docked, set `make_subdirs = 1` to have results for each ligand written to a separate sub-directory.

save_lone_pairs

```
save_lone_pairs = <option>
default: 1 (on)
options: 0 (off) | 1 (on)
```

Some third-party programs have difficulty reading files which contain lone pairs. You can stop GOLD including lone pairs when it writes docked solution files by switching off this option.

bestranking_list_filename

```
bestranking_list_filename = <filename>
```

A file called `bestranking.lst` is written for batch jobs on multiple ligands. This gives a continuous summary of the best solution that has been obtained for each completed ligand. The file contains the total fitness scores and a breakdown of the fitness into its constituent energy terms.

An alternative filename can be specified using the following instruction: `bestranking_list_filename`

fit_points_file

```
fit_points_file = <filename>
default: fit_pts.mol2
```

GOLD automatically calculates a list of hydrophobic fitting points in the binding site. These are used during the generation of trial docking solutions to map hydrophobic ligand atoms into favourable regions of the binding site. GOLD generates its hydrophobic fitting points by placing a fine grid over the binding site. At each grid position, the van der Waals interaction energy between a bare carbon atom and the protein is evaluated. Positions at which the interaction energy is below -2.5 kcal/mol are added to the list of fitting points. In this way, a map is constructed that contains positions onto which the placement of a hydrophobic ligand atom should be favourable. The ligand fitting points are used for the matching of hydrophobic regions

It is possible to instruct GOLD to use customised hydrophobic fitting points (see `read_fitpts`). `fit_points_file` is used to provide GOLD with the location of the customised fitting points, e.g.

```
fit_points_file = Z:\GOLD\datafiles\custom_fit_pts.mol2
```

Customised fitting points must be supplied in a MOL2 format file that contains a list of dummy atoms at the desired fitting-point locations. The supplied fitting points should sample all regions of interest in the cavity, so that the docking algorithm has sufficient alternatives for placement of hydrophobic ligand atoms within the cavity. GOLD uses gridded points that are spaced by 0.25 Å; for a speed-up in calculation, higher values could be used.

Related Instructions:

- read_fitpts

read_fitpts

```
read_fitpts = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

By default, GOLD automatically calculates a list of hydrophobic fitting points in the binding site. These are used during the generation of trial docking solutions to map hydrophobic ligand atoms into favourable regions of the binding site. GOLD generates its hydrophobic fitting points by placing a fine grid over the binding site. At each grid position, the van der Waals interaction energy between a bare carbon atom and the protein is evaluated. Positions at which the interaction energy is below -2.5 kcal/mol are added to the list of fitting points. In this way, a map is constructed that contains positions onto which the placement of a hydrophobic ligand atom should be favourable. The ligand fitting points are used for the matching of hydrophobic regions.

To instruct GOLD to use customised hydrophobic fitting points set `read_fitpts = 1`. `fit_points_file` is used to provide GOLD with the location of the customised fitting points file (see `fit_points_file`).

Related Instructions:

- fit_points_file

Flags

internal_ligand_h_bonds

```
internal_ligand_h_bonds = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

Switching this option on will allow intramolecular hydrogen bonds in the ligand to be formed during docking.

flip_free_corners

```
flip_free_corners = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

Switch this option on to allow free corners of ligand rings to flip. This will result in GOLD performing a limited conformational search of cyclic systems by allowing free corners of rings to flip above or below the plane of their neighbouring atoms. If `flip_free_corners = 0` then rings will be held rigid at the input conformation during docking. The rules governing flipping of ring corners in GOLD are given in: A. W. R. Payne and R. C. Glen, J. Mol. Graphics, 1993, **10**, 74-91

Related Instructions:

- match_ring_templates

match_ring_templates

```
match_ring_templates = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

Switch this option on to allow ring conformational searching using the ring template library. If the ligand contains a ring that is defined in the ring template library the conformation of that ring will vary within the genetic algorithm run. Each time, the current chosen ring conformation changes (it is mutated) and the ligand ring conformation is driven to the newly matched conformation by changing the bond lengths, internal ring angles and torsions.

Related Instructions:

- flip_free_corners

flip_amide_bonds

```
flip_amide_bonds = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

During initialisation of the ligand amides (including thioamides, ureas, and thioureas) will automatically be set to the preferred trans conformation. Setting `flip_amide_bonds = 1` will allow amides, thioamides, ureas, and thioureas in the ligand to subsequently flip between cis and trans during docking.

Note: In order to flip between cis and trans conformations the CO-NRR' torsion is first made planar (at the initialised trans conformation). N,N disubstituted amides are not made planar; CO-NH_2 will be set so that the NH_2 group is in plane with the CO (care must be taken that the input RNH_2 group itself is planar since GOLD will not change this).

Related Instructions:

- postprocess_bonds
- rotatable_bond_override_file

flip_planar_n

```
flip_planar_n = <option> <flip_ring_NRR | fix_ring_NRR |
rot_ring_NRR> <flip_ring_NHR | fix_ring_NHR | rot_ring_NHR>

default: 1 flip_ring_NRR flip_ring_NHR
options: 0 (off) | 1 (on)
```

Set `flip_planar_n = 1` to allow planar trigonal nitrogens in the ligand (bound to sp^2 carbons) to flip between cis and trans conformations during docking (otherwise, they will be held fixed at the input geometry).

In addition, it is possible to specifically control the behaviour of ring-NHR and ring-NRR' groups by using the following keywords:

- `flip_ring_NRR`: use to allow NRR' groups to flip (i.e. rotate 180 degrees) during docking.
- `fix_ring_NRR`: use to fix NRR' groups at their input conformation.
- `rot_ring_NRR`: use to allow free rotation of NRR' groups.
- `flip_ring_NHR`: use to allow NHR groups to flip (i.e. rotate 180 degrees) during docking.
- `fix_ring_NHR`: use to fix NHR groups at their input conformation.
- `rot_ring_NHR`: use to allow free rotation of NHR groups.

For example, setting `flip_planar_n = 1 fix_ring_NRR` will allow all planar R_3N groups to flip, but will fix ring-NRR' groups at their input conformation.

Related Instructions:

- `postprocess_bonds`
- `rotatable_bond_override_file`

flip_pyramidal_n

```
flip_pyramidal_n = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

Switch this option on to allow pyramidal (i.e. non-planar sp^3) nitrogens to invert during docking (otherwise, they will be held fixed at the input geometry). Given a non-planar group RR'R"N or tetrahedrally surrounded RR'R"NH, setting `flip_pyramidal_n = 1` will enable flipping of the local stereochemistry around the nitrogen (the energy barrier for this umbrella-like change of geometry around the nitrogen is low). Flipping only changes the stereochemistry around RR'R"N and RR'R"NH nitrogens. It does not affect other chiral centres.

rotate_carboxylic_oh

```
rotate_carboxylic_oh = < flip | rotate | fix >
default: fix
```

The instruction `rotate_carboxylic_oh` can be used to control the behaviour of protonated carboxylic acids during docking. The possible settings are:

- `flip`: protonated carboxylic acids will be allowed to flip (i.e. rotate 180 degrees) during docking.
- `rotate`: protonated carboxylic acids will be allowed to rotate freely during docking.
- `fix`: protonated carboxylic acids will be held rigid at their input conformation.

use_tordist

```
use_tordist = <option>
default: 1 (on)
options: 0 (off) | 1 (on)
```

Torsion angle distributions extracted from the Cambridge Structural Database (CSD) can be input to GOLD. When `use_tordist` is switched on these distributions are used to restrict the ligand conformational space sampled by the genetic algorithm. Using torsion angle distributions in this way may improve the chances of GOLD finding the correct answer by biasing the search towards ligand torsion-angle values that are commonly observed in crystal structures.

The instruction `tordist_file` is used to provide GOLD with the location of the torsion angle distribution file that is to be used (see `tordist_file`).

Related Instructions:

- `tordist_file`

fix_rotatable bond

```
fix_rotatable_bond = < <atom number 1> <atom number 2> | all
| all_but_terminal >
```

GOLD was designed to dock flexible ligands into protein binding sites. However, sometimes it can be useful to fix the geometry of specific bonds or of part of the ligand, e.g. in order to study the possible binding of a pre-determined ligand geometry. The following options are available:

- Use `fix_rotatable_bond = <atom number 1> <atom number 2>` to fix a single rotatable bond between two atoms. The atom ids as they appear in the ligand input file must be specified.
- Use `fix_rotatable bond = all` to fix all rotatable bonds in the ligand at their input conformation.
- Use `fix_rotatable_bond = all_but_terminal` to fix all non-terminal rotatable bonds (i.e. not -CH_3, -OH, etc.), at their input conformation.

Note: When fixing all rotatable bonds at their input conformation (i.e. performing a rigid ligand docking) GOLD will not perform a local optimisation (simplex) on the final solution. This may lead to penalisation of near-optimal conformations.

postprocess_bonds

```
postprocess_bonds = <option>
default: 1 (on)
options: 0 (off) | 1 (on)
```

By default, certain bonds are treated in a specific manner at ligand initialisation in order to prepare them for docking. However, if a bond is e.g. desired to rotate freely rather than flip during docking, then fine-grained control over specific substructures can be achieved by post processing bonds after ligand initialisation.

Set `postprocess_bonds = 1` to allow post-processing of bonds. Control over specific substrucures can then be achieved by using the `rotatable_bond_override.mol2` file, found in the `$GOLD_DIR/gold/` directory (see `rotatable_bond_override_file`).

Related Instructions:

- `rotatable_bond_override_file`

rotatable_bond_override_file

```
rotatable_bond_override_file = <filename>
default: $GOLD_DIR\gold\rotatable_bond_override.mol2
```

By default, certain bonds are treated in a specific manner at ligand initialisation in order to prepare them for docking. However, control over specific substructures can be achieved by post-processing bonds after ligand initialisation.

`postprocess_bonds = 1` (see `postprocess_bonds`) must be set to allow post-processing of bonds. Fine-grained control can then be achieved over specific substructures by using the `rotatable_bond_override.mol2` file, found in the `$GOLD_DIR/gold/` directory. Some fragments are already provided (which can be edited), however user-specific ones may also be added. Instructions on how to do this, as well as further information, can be found in the file itself. This is useful if further control is sought over more than one ligand with a common substructure in a ligand library file.

`rotatable_bond_override_file` is used to provide GOLD with the location (i.e. full path and filename) of the `rotatable_bond_override.mol2` file. The file itself can be customised by copying it, editing the copy, and instructing GOLD to use the edited file.

Related Instructions:

- `postprocess_bonds`

diverse_solutions

```
diverse_solutions = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

This option allows control over whether or not diverse solutions are generated for a docking run. Note: If diverse solutions have never been generated for a particular GOLD run, the `diverse_solutions = 0` line may not be present in the `gold.conf`.

The `diverse_solutions` instruction in the `gold.conf` should be accompanied by `divsol_cluster_size` and `divsol_rmsd` instructions.

Related instructions:

- divsol_cluster_size
- divsol_rmsd

divsol_cluster_size

```
divsol_cluster_size = <value>
default: 1
```

When generating diverse docking solutions, use the `divsol_cluster_size` tag to specify how many ligand diverse solutions are contained in a cluster.

Related instructions:

- diverse_solutions
- divsol_rmsd

divsol_rmsd

```
divsol_rmsd = <value>
default: 1.5
```

When generating diverse docking solutions, use this setting to define the RMSD cut-off (in Å) for determining if diverse solutions are in the same cluster or not.

Related instructions:

- diverse_solutions

solvate_all

```
solvate_all = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

When the binding site is generated potential donor and acceptor fitting points are added to solvent accessible atoms, the potential fitting points are themselves tested for solvent accessibility, and only those fitting points that are accessible are used. It is possible to remove this requirement for fitting points to be solvent accessible. In this case fitting points would be generated for all solvent accessible donor and acceptor atoms within the binding site. Note that these atoms are already deemed to be solvent accessible but it's their potential fitting points that may have been desolvated by neighbouring atoms. This option can be used e.g. to avoid problems with solvent accessibility of backbone carbonyls in kinases where one of the carbonyl lone pairs is typically desolvated by a neighbouring atom. To generate fitting points for all solvent accessible donor and acceptor atoms set `solvate_all = 1`, the default is 0 (off).

Related Instructions:

- radius
- origin
- do_cavity
- floodfill_atom_no
- cavity_file
- floodfill_center

fix_all_protein_rotatable_bonds

```
fix_all_protein_rotatable_bonds = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

During all dockings serine, threonine and tyrosine hydroxyl groups are optimised, i.e. rotated during docking, as are lysine NH_3^+ groups. If this is undesirable this rotation can be switched off by setting `fix_all_protein_rotatable_bonds = 1`.

Termination

early_termination

```
early_termination = <option>
default: 1 (on)
options: 0 (off) | 1 (on)
```

Setting `early_termination = 1` will instruct GOLD to terminate docking runs on a given ligand as soon as a specified number of runs have given essentially the same answer. In this situation, it is probable that the answer is correct, and GOLD will just be wasting time if it performs more docking runs on that ligand.

The early termination criterion must also be specified. GOLD will stop docking a ligand when a specified number of top solutions (see `n_top_solutions`) are all within a specified RMSD (see `rms_tolerance`) of each other.

Related Instructions:

- `n_top_solutions`
- `rms_tolerance`

n_top_solutions

```
n_top_solutions = <value>
default: 3
```

GOLD can be instructed to terminate docking runs on a given ligand as soon as a specified number of runs have given essentially the same answer (see early_termination). In this situation, it is probable that the answer is correct, and GOLD will just be wasting time if it performs more docking runs on that ligand.

The early termination criterion must be specified. GOLD will stop docking a ligand when the specified number of top solutions <value> are all within a specified RMSD (see rms_tolerance) of each other.

Related Instructions:

- early_termination
- rms_tolerance

rms_tolerance

```
rms_tolerance = <value>
default: 1.5
```

GOLD can be instructed to terminate docking runs on a given ligand as soon as a specified number of runs have given essentially the same answer (see early_termination). In this situation, it is probable that the answer is correct, and GOLD will just be wasting time if it performs more docking runs on that ligand.

The early termination criterion must be specified. GOLD will stop docking a ligand when the specified number of top solutions (see n_top_solutions) are all within <value> Å RMSD of each other.

Related Instructions:

- early_termination
- n_top_solutions

Constraints

constraint distance

```
constraint distance <protein | ligand> <atom id> <protein |
ligand> <atom id> <max. distance> <min. distance> <spring
constant> <on | off>
```

A distance between a specified ligand and protein atom (or between two ligand atoms) can be constrained to lie between minimum and maximum distance bounds. During a GOLD run, if a constrained distance is found to lie outside its bounds, a spring energy term is used to reduce the fitness score, i.e.

$E = kx^2$

where: x is the difference between the distance and the closest constraint bound; k is a user-defined spring constant.

When using a distance constraint GOLD will therefore be biased towards finding solutions in which the specified constraint is satisfied. However, it is important to remember that such a solution is not guaranteed (i.e. it is not possible to force a constraint to be satisfied in the final solution). It is possible to instruct GOLD not to dock ligands when the specified constraint(s) are physically impossible to satisfy (e.g. if no suitable group is present in the ligand to form the required constraint) (see `force_constraints`).

For each atom involved in the constraint it is necessary to specify whether the atom belongs to the protein or the ligand file `<protein | ligand >`. The atom id (as it appears in the structure input file) must also be specified. The minimum `<min. distance>` and maximum `<max. distance>` separation of the constrained atoms must be entered (distances are in Å), and the spring constant `<spring constant>` must also be specified (by default, this is set to 5.0).

If an atom specified in the constraint is topologically equivalent to other atoms (e.g. it is one of the oxygen atoms of an ionised carboxylate group), then it is possible to instruct GOLD to

automatically compute the constraint term using whichever of the equivalent atoms gives the best value. Use `<on | off>` to control whether or not topologically equivalent atoms are considered.

For example, the following instruction defines a constraint between a zinc atom in the protein (atom id 2041) and a sulfonamide nitrogen in the ligand (atom id 8). A maximum separation of 3.50 Å and a minimum separation of 1.50 Å have been specified and a spring constant of 5.0 is used:

```
constraint distance protein 2041 ligand 8 1.5 3.5 5.0 off
```

Related Instructions:

- `force_constraints`
- `constraint substructure`

constraint h_bond

```
constraint h_bond ligand <atom id> protein <atom id>
```

A ligand atom may be constrained to form a hydrogen bond to a particular protein atom. One atom should be a donatable hydrogen atom (you must give the atom id of the hydrogen atom, not the O or N atom to which it is attached) and the other should be an acceptor. The atom ids (as they appear in the structure input files) must be specified. The protein atom should be available for ligand binding (i.e. solvent accessible).

The constraint is incorporated into the least-squares fitting routine used by GOLD. Thus, when least-squares fitting is used to dock the ligand (by attempting to form hydrogen bonds encoded within the chromosome) the constraint is added to the least-squares mapping. The constraint has a weight of 5 relative to a normal hydrogen bond taken from the chromosome.

The hydrogen bond constraint weighting can be altered within the `# FITNESS FUNCTION` section of the GOLD parameters file by changing the value of the parameter `CONSTRAINT_WT`.

When using constraints GOLD will be biased towards finding solutions in which the specified constraint is satisfied. However, it is important to remember that such a solution is not guaranteed (i.e. it is not possible to force a constraint to be satisfied in the final solution). It is possible to instruct GOLD not to dock ligands when the specified constraint(s) are physically impossible to satisfy (e.g. if no suitable group is present in the ligand to form the required constraint) (see `force_constraints`).

For example, the following instruction defines a constraint between a carboxylate oxygen (atom id 3401) of an aspartate residue in the protein and donatable hydrogen (atom id 17) in the ligand.

```
constraint h_bond ligand 17 protein 3401
```

Related Instructions:

- `force_constraints`
- `constraint protein_h_bond`

constraint protein_h_bond

```
constraint h_bond <constraint weight> <min. geometry weight>
<atom id>
```

A protein hydrogen bond constraint can be used to specify that a particular protein atom should be hydrogen-bonded to the ligand, but without specifying to which ligand atom.

GOLD will be biased towards finding solutions in which the specified protein atom(s) form hydrogen bonds. The fitness score of a given docking will be penalised for every protein H-bond constraint that is not satisfied (i.e. for every protein atom that you have specified should form a hydrogen bond but does not). The magnitude of this penalty is equal to the `<constraint weight>` that is specified. The `<constraint weight>` is also the strength of bias applied to the formation of a specified hydrogen bond in the least squares mapping algorithm within GOLD.

The `<min. geometry weight>` is a user defined value that determines how good a hydrogen bonding interaction has to be in order for it to be considered a hydrogen bond by GOLD. The minimum H-bond geometry weight takes a range of values from 0 to 1, by default this value is set at 0.005. During docking GOLD assesses the geometry of each required hydrogen bond on a scale of 0 to 1, with 1 denoting perfect. If this geometry weight for the constrained H-bond falls below the minimum H-bond geometry weight specified, a penalty `<constraint weight>` will be applied to the score for the unfulfilled hydrogen bond; i.e. it will not be considered to be an H-bond and will therefore contribute a penalty to the fitness score.

The protein atom(s) to be constrained should be specified using `<atom ids>`. The atom indice(s) as defined in the ligand input file must be used. Either a donatable hydrogen atom (you must give the number of the hydrogen atom, not the O or N atom to which it is attached) or an acceptor can be specified. The protein atom(s) should also be available for ligand binding (i.e. solvent accessible).

For a given protein H bond constraint more than one protein atom number can be specified. This will instruct GOLD to use an either-or type of constraint during docking. For example, specifying two protein atoms, acceptor m and acceptor n, separated by a space, will result in the constraint being satisfied if an H bond is formed to either m or n during docking. This is of use when defining constraints involving, for example, carboxylates where it is not important which oxygen atom forms an H bond, provided that one of them does.

When using constraints GOLD will be biased towards finding solutions in which the specified constraint is satisfied. However, it is important to remember that such a solution is not guaranteed (i.e. it is not possible to force a constraint to be satisfied in the final solution). It is possible to instruct GOLD not to dock ligands when the specified constraint(s) are physically impossible to satisfy (e.g. if no suitable group is present in the ligand to form the required constraint) (see `force_constraints`).

For example, the following instruction defines a constraint where either oxygen atom (atom ids 241 and 242) of a carboxylate group should form a hydrogen bond to the ligand. A constraint weight of

10.0 has been specified, and a minimum geometry weight of 0.005 is used. The constraint will be satisfied if either of the carboxylate oxygen atoms forms the required hydrogen bond.

```
constraint protein_h_bond 10.000000 0.005000 241 242
```

Related Instructions:

- constraint h_bond
- force_constraints

constraint sphere

```
constraint sphere <x value> <y value> <z value> <radius>
<score> <hydrophobic atoms | arom_ring_atoms | list <atom
ids>>
```

This constraint can be used to bias the docking towards solutions in which particular regions of the binding site (e.g. a hydrophobic pocket) are occupied by specific ligand atoms (or types of ligand atom).

For each constraint specified a sphere is placed at an explicitly-defined position (using x,y,z coordinates) within the binding site. Each sphere is assigned a user-defined radius, so a sphere can be adjusted if required, e.g. to fill an entire pocket in the binding-site.

A contribution (determined according to a user-specified weighting) is then added to the score for each specified non-hydrogen ligand atom that lies within the designated sphere. Note: the contribution is added to the score for each atom located within the sphere, i.e. the total contribution will depend on the number of atoms found in the region of interest and ultimately the ligand-accessible volume of the region.

`<x value> <y value> <z value>` are used to specify the position of the sphere within the binding site. The sphere must also be assigned a radius `<radius>`. The minimum settable value of `<radius>` is 0.5 Å.

The value of `<score>` is the contribution that will be added to the score for each specified non-hydrogen ligand atom that lies within the designated sphere.

The ligand atoms used in the constraint can be specified explicitly from a list of atom numbers. Atoms should be specified using `list <atom ids>`. The atom indices as defined in the ligand input file must be used. Atom indices should be separated by a single space. Alternatively, it is possible to use all hydrophobic ligand atoms, or to use only those hydrophobic atoms in aromatic rings: `<hydrophobic atoms | arom_ring_atoms>`.

Atoms considered to be hydrophobic include:

- Carbon atoms bound to at least two H or C atoms.
- Atoms typed C.cat.
- Atoms typed S.3 and bound to two carbons.
- H atoms bound to an sp^2, sp^3 or aromatic carbon (Note: only heavy atoms found within the sphere will contribute to the score).

When using constraints GOLD will be biased towards finding solutions in which the specified constraint is satisfied. However, it is important to remember that such a solution is not guaranteed (i.e. it is not possible to force a constraint to be satisfied in the final solution). It is possible to instruct GOLD not to dock ligands when the specified constraint(s) are physically impossible to satisfy (e.g. if no suitable group is present in the ligand to form the required constraint) (see `force_constraints`).

Related Instructions:

- `force_constraints`

constraint pharmacophore

```
constraint pharmacophore <DON | ACC | D_A| RING_CNT | ARO_CNT
> <BLOCK | GAUSS> <x value> <y value> <z value> <0 | 1>
<radius> <score> <fit point weight>
```

This constraint can be used to bias the docking towards solutions in which particular regions of the binding site are occupied by specific types of ligand atom (i.e. hydrogen-bond donor `<DON>`, hydrogen-bond acceptor `<ACC>`, hydrogen-bond donor or acceptor `<D_A>`, ring centre `<RING_CNT>` or aromatic ring centre `<RING_CNT>`).

For each constraint specified a sphere is placed at an explicitly-defined position (using x,y,z coordinates) within the binding site. Each sphere is assigned a user-defined radius, so a sphere can be adjusted if required.

A contribution (determined according to a user-specified weighting) is then added to the score for each specified pharmacophore point. A contribution is added to the score if the distance between the pharmacophore point and an atom of the same type in the molecule is less than the specified radius of the sphere. If a pharmacophore constraint point is not matched in the molecule, it will not be used during scoring.

`<x value> <y value> <z value>` are used to specify the position of the sphere within the binding site. The sphere must also be assigned a radius `<radius>`.

`<0|1>` It is possible to add these pharmacophore points to the list of fitting points used by GOLD. This value sets whether to use this point as a fitting point in the mapping algorithm (1 is on).

The value of `<score>` is the contribution that will be added to the score for each matched pharmacophore point.

The value of `<fit point weight>` is the weight that will be used for the pharmacophore point in the fitting algorithm. A high weight will effectively force dockings to match if possible.

constraint substructure

```
constraint substructure protein <atom id> <filename> <atom
id> <max. distance> <min. distance> <spring constant>
ring_center
```

It is possible to apply a distance constraint to multiple ligands which have a common functional group, or fragment. The constraint will bias the distance between a protein atom and one atom of the fragment towards a specified distance range.

During a GOLD run, if a constrained distance is found to lie outside its bounds, a spring energy term is used to reduce the fitness score, i.e.

$E = kx^2$

where: x is the difference between the distance and the closest constraint bound; k is a user-defined spring constant.

During docking the constraint will be applied to any ligands which contain the specified fragment (matching is performed on the basis of the atom types and 2D connectivity) and the resulting solutions will be biased towards the specified distance range.

`<filename>` is used to provide GOLD with the location of the fragment file. The fragment must be supplied as a MOL2 file or PDB file.

The protein atom number and fragment atom number must be specified. The atom ids (as they appear in the structure input file) must be used. The minimum `<min. distance>` and maximum `<max. distance>` separation of the constrained atoms must be entered (distances are in Å), and the spring constant `<spring constant>` must also be specified (by default, this is set to 5.0).

It is possible to define a distance constraint from a centroid of a ring in the ligand. To do this, specify a fragment atom within the ring of interest and use the keyword `ring_center`. The closest ring centre to the selected atom will be used. Note: When defining a distance constraint involving a ring centre, ensure that the maximum and minimum separations are adjusted accordingly.

If the constraint refers to a fragment atom (and therefore a ligand atom) which is topologically equivalent to other atoms (e.g. it is one of the oxygen atoms of a carboxylate group), GOLD will automatically compute the constraint term using whichever of the equivalent atoms gives the best value.

When using constraints GOLD will be biased towards finding solutions in which the specified constraint is satisfied. However, it is important to remember that such a solution is not guaranteed (i.e. it is not possible to force a constraint to be satisfied in the final solution). It is possible to instruct GOLD not to dock ligands when the specified constraint(s) are physically impossible to satisfy (e.g. if no suitable group is present in the ligand to form the required constraint) (see `force_constraints`).

Related Instructions:

- `constraint distance`
- `force_constraints`

constraint similarity

```
constraint similarity <donor | acceptor | all> <filename> <weight>
```

This constraint will bias the conformation of docked ligands towards a given solution. This solution, or template, can, for example, be another ligand in a known conformation, a common core, or it may just be a large substructure that is expected, or known, to bind in a certain way.

An energy term will be added to the score based on the similarity between the ligand being docked and the template provided. The similarity between the two is evaluated as a Gaussian overlap term.

The template file should contain the template molecule or fragment in its docked position (i.e. expressed with respect to the same coordinate frame as the protein and with the coordinates required to place it in the correct pose). `<filename>` is used to provide GOLD with the location of the template file. The template must be supplied as a MOL2 file or PDB file.

The similarity constraint can be applied in three ways that differ in the way that the overlap between ligand and template is calculated. The similarity can be evaluated:

- by using the overlap between all donor atoms in the template and the ligand being docked: `constraint similarity donor <filename> <weight>`
- by using the overlap between all acceptor atoms in the template and the ligand being docked: `constraint similarity acceptor <filename> <weight>`
- by using the overlap of all atoms of the template (this can be regarded as a ligand-shape constraint): `constraint similarity all <filename> <weight>`

The value of `<weight>` determines the maximum energy term that would be added to the score in the case of perfect overlap between ligand and template. The energy term to be added is calculated as similarity times weight (the similarity value is between 0 and 1, where 1 indicates an identical match between template and ligand). As an initial value for this term, we suggest a value between 5 and 30.

When using constraints GOLD will be biased towards finding solutions in which the specified constraint is satisfied. However, it is important to remember that such a solution is not guaranteed (i.e. it is not possible to force a constraint to be satisfied in the final solution). It is possible to instruct GOLD not to dock ligands when the specified constraint(s) are physically impossible to satisfy (e.g. if no suitable group is present in the ligand to form the required constraint) (see `force_constraints`).

Related Instructions:

- `force_constraints`

constraint scaffold

```
constraint scaffold <filename> <weight> list <atom ids>
```

The scaffold match constraint can be used to place a fragment at an exact specified position in the binding site, the geometry of the fragment will not be altered during docking. The scaffold can, for example, be a common core, or fragment, or it may just be a substructure known to adopt a certain binding position.

The constraint is enforced at the mapping stage in GOLD. Ligand placements are generated using a best least-squares fit with the scaffold heavy atom positions, i.e. this constraint forces all atoms on the matching portion of the ligand to lie very close to, or coincident with, the corresponding scaffold. There is no S(con) contribution to the fitness score to bias dockings.

The scaffold file should contain the scaffold fragment in its docked position (i.e. expressed in the same coordinate frame as the protein and with the coordinates required to place it in the correct pose). `<filename>` is used to provide GOLD with the location of the scaffold file. The scaffold must be supplied as a MOL2 file.

The value of `<weight>` determines how closely ligand atoms fit onto the scaffold. Setting a higher weight will force the ligand to be placed onto the scaffold locations more strictly. A default weight of 5.0 is used. Values below 1 can be used to achieve a more lenient overlay.

By default, all heavy atoms in the supplied scaffold structure file will be used for matching. However, it is possible to specify only a subset of those atoms in the scaffold structure (these may include non-heavy atoms). Atoms should be specified using `list <atom ids>`. The atom indices as defined in the scaffold structure file must be used. Atom indices should be separated by a single space.

Related Instructions:

- `force_constraints`

interaction_restraint_weight

```
interaction_restraint_weight = <value>
default: 50
```

When using the `constrain interaction_restraint` option the docking contribution added to the fitness score of ligand poses in which a motif is matched (i.e. poses in which all the interactions defined as part of a motif are satisfied) is based upon the accumulated hydrogen-bonding and lipophilic interactions defined as part of that motif and the `interaction_restraint_weight`.

The `interaction_restraint_weight` can be used to customise the overall importance of the `constraint interaction_restraint`.

Related Instructions:

- `constraint interaction_restraint`

constraint interaction_restraint

```
constraint interaction_restraint <interaction type> <residue number> <chain id> <residue name> <atom name> <motif 1> <motif 2> ... <motif n>
```

The interaction motif constraint can be used to bias the docking towards solutions that form particular motifs of interactions.

During docking a contribution will be added to the fitness score of ligand poses in which a motif is matched (i.e. poses in which all the interactions defined as part of a motif are satisfied). This contribution is based upon the accumulated hydrogen-bonding and lipophilic interactions defined as part of that motif and the `interaction_restraint_weight` (see interaction_restraint_weight). Therefore, docking will be biased towards ligand poses which form interactions to the protein atoms of interest matching one of the uniquely defined motifs.

Typically several lines of the gold.conf file would be used to describe all the interactions of interest in the interaction motif using the format outlined above.

The available `<interaction type>` are: H-bond acceptor (1); H-bond donor (2); lipophilic interaction (3); CHO donor (4).

The values `<residue number> <chain id> <residue name> <atom name>` are used to identify the protein atom that forms the interaction.

In the case of H-bond donors, H-bond acceptors and CHO donors the values for `<motif 1> <motif 2> ... <motif n>` are set to either 1 or 0 depending on whether or not the interaction is observed in that particular motif or not. For the lipophilic interactions the values are set to the frequency of that interaction in a set of complexes and are added to all the motifs.

For example, the following instruction defines an interaction motif consisting of 11 motifs. Each line represents a unique interaction that the protein can form (i.e. the first two lines define that the carbonyl group of GLU81 can act as either a hydrogen bond donor or a CHO donor). Columns 8 to 18 represent unique interaction motifs.

```
constraint interaction_restraint 1 81 A GLU O 0 1 1 1 0 0 0 1 0 0
        1
constraint interaction_restraint 4 81 A GLU O 0 0 0 0 1 0 0 0 0 1
        0
constraint interaction_restraint 2 83 A LEU N 1 1 1 0 1 1 0 1 0 0
        0
constraint interaction_restraint 1 83 A LEU O 1 0 0 0 1 1 1 1 1 1
        1
constraint interaction_restraint 4 83 A LEU O 0 0 1 0 0 0 0 0 0 0
        0
constraint interaction_restraint 3 31 A ALA CB 1 1 1 1 1 1 1 1 1
        1 1
constraint interaction_restraint 3 134 A LEU CD1 1 1 1 1 1 1 1 1
        1 1 1
constraint interaction_restraint 3 80 A PHE CB 0.59 0.59 0.59
        0.59 0.59
0.59 0.59 0.59 0.59 0.59 0.59
```

Related Instructions:

- interaction_restraint_weight

force_constraints

```
force_constraints = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

Setting `force_constraints = 1` will instruct GOLD not to dock ligands when the specified constraint(s) are physically impossible to satisfy (e.g. if no suitable group is present in the ligand to form the required H-bond constraint).

Covalent Bonding

covalent

```
covalent = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

Set `covalent = 1` in order to dock covalently bound ligands.

GOLD assumes that there is just one atom linking the ligand to the protein (e.g. the O in a serine residue). Both protein and ligand files should be set up with the link atom included (so, if the serine O is the link atom, it will appear in both the protein and ligand input files).

Ideally the link atom, in both the ligand and the protein, will have a free valence available through which the link can be made. If the link atom on the ligand does not have a free valence, having a hydrogen instead, then the docking will proceed and the hydrogen will be ignored in terms of its contribution to the fitness score. It will however still be displayed when docking poses are visualised.

It is necessary to specify which ligand atom (see `covalent_ligand_atom_no`) is bonded to which protein atom (see `covalent_protein_atom_no`) when setting up the docking.

Inside the GOLD least-squares fitting routine, the link atom in the ligand will be forced to fit onto the link atom in the protein.

In order to make sure that the geometry of the bound ligand is correct, the angle-bending potential from the Tripos Force Field has been incorporated into the fitness function. On evaluating the score for the docked ligand, the angle-bending energy for the link atom is included in the calculation of the fitness score.

Related Instructions:

- covalent_protein_atom_no
- covalent_ligand_atom_no
- covalent_substructure_filename
- covalent_substructure_atom_no
- covalent_substructure
- covalent_topology

covalent_protein_atom_no

`covalent_protein_atom_no = <atom id>`

When docking covalently bound ligands (see covalent) GOLD will assume that there is just one atom linking the ligand to the protein (e.g. the O in a serine residue). Both protein and ligand files should be set up with the link atom included.

`covalent_protein_atom_no` is used to define the link atom in the protein file. The atom id (as it appears in the protein input file) must be specified.

The link atom as it appears in the ligand input file (see covalent_ligand_atom_no), or the substructure file (for use with multiple ligands which have a common functional group) (see covalent_substructure) must also be specified.

Related Instructions:

- covalent
- covalent_ligand_atom_no

- covalent_substructure_filename
- covalent_substructure_atom_no
- covalent_substructure
- covalent_topology

covalent_ligand_atom_no

`covalent_ligand_atom_no = <atom id>`

When docking covalently bound ligands (see covalent) GOLD will assume that there is just one atom linking the ligand to the protein (e.g. the O in a serine residue). Both protein and ligand files should be set up with the link atom included.

`covalent_ligand_atom_no` is used to define the link atom in the ligand file. The atom id (as it appears in the ligand input file) must be specified.

The link atom as it appears in the protein input file must also be specified (see covalent_protein_atom_no).

GOLD supports two types of covalent link: a covalent link for use with individual ligands, and a substructure-based covalent link for use with multiple ligands which have a common functional group (see covalent_substructure). `covalent_ligand_atom_no` should only be set when defining a covalent link to an individual ligand.

Related Instructions:

- covalent
- covalent_protein_atom_no

covalent_substructure_filename

`covalent_substructure_filename = <filename>`

It is possible to apply a covalent link to multiple ligands which have a common functional group (see covalent_substructure). During docking the link will be applied to any ligands which contain a specified substructure.

To use a substructure-based covalent link, first create a file containing the substructure in MOL2 format (e.g. `substructure.mol2`). It is recommended that you set atom types manually since an incomplete fragment can cause problems with automatic atom-typing. The actual conformation of the group in this file is not important, as only the atom types and 2D connectivity will be used for matching.

`covalent_substructure_filename` is used to provide GOLD with the location of the substructure file and must be followed by a filename.

The substructure atom number to which the covalent link applies must also be specified (see `covalent_substructure_atom_no`).

Related Instructions:

- `covalent`
- `covalent_ligand_atom_no`
- `covalent_substructure`
- `covalent_substructure_atom_no`
- `covalent_topology`

covalent_substructure_atom_no

`covalent_substructure_atom_no = <atom id>`

It is possible to apply a covalent link to multiple ligands which have a common functional group (see `covalent_substructure`). During docking the link will be applied to any ligands which contain a specified substructure (see `covalent_substructure_filename`).

`covalent_substructure_atom_no` is used to define the substructure atom number to which the covalent link applies. The atom id (as it appears in the substructure input file) must be specified.

Related Instructions:

- `covalent`
- `covalent_protein_atom_no`
- `covalent_substructure_filename`
- `covalent_substructure`
- `covalent_topology`

covalent_substructure

```
covalent_substructure = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

GOLD supports two types of covalent link: a covalent link for use with individual ligands, and a substructure-based covalent link for use with multiple ligands which have a common functional group.

Set `covalent_substructure = 1` to define a substructure-based covalent link. During docking the link will then be applied to any ligands which contain the specified substructure (see `covalent_substructure_filename`).

Related Instructions:

- `covalent`
- `covalent_protein_atom_no`
- `covalent_substructure_filename`
- `covalent_substructure_atom_no`
- `covalent_topology`

covalent_topology

```
covalent_topology = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

When setting up a substructure-based covalent link (see covalent_substructure) in which the specified substructure atom (and therefore ligand atom) is topologically equivalent to other atoms (e.g. it is one of the oxygen atoms of a carboxylate group), it is possible to instruct GOLD to use whichever of the equivalent atoms gives the best result.

Set `covalent_topology = 1` to consider topologically equivalent ligand atoms.

Related instructions

- covalent
- covalent_protein_atom_no
- covalent_substructure_filename
- covalent_substructure_atom_no
- covalent_substructure

Save Options

save_score_in_file

```
save_score_in_file = <option> [weighted | unweighted | all
[no_sdtags_in_mol2] [comments]]

default: 1 (on)
options: 0 (off) | 1 (on)
```

It is possible to write additional information to docked solution files. This information is written to SD file tags; for MOL2 files, these tags are written to comment blocks. This information can be used in the post-processing of docking results.

Set `save_score_in_file = 1` in order to include the docking-score terms (i.e. the total GoldScore or ChemScore value for each docking, and its components such as protein-ligand H-bond energy, internal ligand strain energy, etc.) in the docked solution files.

Certain docking scoring function terms are the product of a term dependent on the magnitude of a particular physical contribution (e.g. hydrogen bonding) and a scale factor determined e.g. by a regression coefficient. The docking scoring function terms included in the output file can therefore consist of weighted terms, non-weighted terms or both.

To include weighted scoring terms only specify the keyword `weighted`, to include non-weighted terms only specify the keyword `unweighted`, or to include both weighted and non-weighted terms specify the keyword `all`.

To prevent SD-style tags being written to comment blocks in MOL2 solution files, specify the keyword `no_sdtags_in_mol2`.

If your input file contained MOL2 file tags, then these can be preserved in a MOL2 comment field in the output file by specifying the keyword `comments`.

save_protein_torsions

```
save_protein_torsions = <option>
default: 1 (on)
options: 0 (off) | 1 (on)
```

It is possible to specify that one or more protein side chains are to be treated as flexible (see `rotamer_lib`). Each flexible side chain will be allowed to undergo torsional rotation around one or more of its acyclic bonds during docking.

In addition, the torsion angles of Ser, Thr and Tyr hydroxyl groups in the protein will be automatically optimised by GOLD. Specifically, each Ser, Thr and Tyr OH will be allowed to rotate to optimise its hydrogen-bonding to the ligand. Lysine NH_3^+ groups are similarly optimised.

These optimised protein torsions that are generated during docking (these will usually be different for each docked ligand pose) can be written to docked solution files. This information is written to SD file tags; for MOL2 files, these tags are written to comment blocks.

concatenated_output

`concatenated_output = <filename>`

By default, the result of each docking attempt is written out to a separate file `gold_soln_structure_m#_n.mol2`, where `n` is the solution number 1,2,3 ... and `m#` is the number of the ligand, i.e. `m1` for the first ligand in the input file, `m2` for the second, etc.

Alternatively, it is possible to specify that all saved docking solutions for all ligands are to be concatenated and written to a single file (in MOL2 or SD format).

`concatenated_output` is used to provide GOLD with the location of the file to which all solutions are to be written. `concatenated_output` must be followed by a filename.

Note: When performing a rescoring run (see `run_flag`), GOLD will, by default, write out docked ligand solutions after rescoring. Solutions will be written to the file `rescore.mol2`. A log file, `rescore.log`, which summarises the outcome of the rescoring run will also be written. An alternative filename (for both the rescore solution and log files), can also be specified using the `concatenated_output` instruction.

Related Instructions:

- `clean_up_option delete_all_solutions`
- `clean_up_option delete_redundant_log_files`

- clean_up_option delete_empty_directories
- run_flag

clean_up_option delete_all_solutions

`clean_up_option delete_all_solutions`

By default, the result of each docking attempt is written out to a separate file `gold_soln_structure_m#_n.mol2`, where `n` is the solution number 1,2,3 ... and `m#` is the number of the ligand, i.e. `m1` for the first ligand in the input file, `m2` for the second, etc.

If you do not wish to retain these individual solution files, e.g. when writing all solutions to a single concatenated file (see `concatenated_output`), then specify `clean_up_option delete_all_solutions` in order to delete the unwanted solution files.

Related Instructions:

- concatenated_output
- clean_up_option delete_redundant_log_files
- clean_up_option delete_all_initialised_ligands
- clean_up_option delete_empty_directories

clean_up_option save_fitness_better_than

`clean_up_option save_fitness_better_than <value>`

Set `clean_up_option save_fitness_better_than` in order to filter out all solutions with fitness scores lower than `<value>`, i.e. solutions with fitness scores lower than that specified will be rejected. For example, setting `clean_up_option save_fitness_better_than 50` will mean that any solution with a fitness lower than 50 will not be kept.

Related Instructions

- clean_up_option save_top_n_solutions
- clean_up_option save_best_ligands

clean_up_option save_top_n_solutions

`clean_up_option save_top_n_solutions <value>`

By default, all docked solution will be kept at the end of a docking run. However, GOLD can produce a lot of output and you may wish to cut this down.

Set `clean_up_option save_top_n_solutions` in order to retain just the `<value>` best solutions for each ligand.

In order for `clean_up_option save_top_n_solutions` to take effect the options `clean_up_option delete_empty_directories` and `clean_up_option delete_redundant_log_files` also need to be set.

Related Instructions

- `clean_up_option delete_empty_directories`
- `clean_up_option delete_redundant_log_files`
- `clean_up_option save_fitness_better_than`

clean_up_option save_best_ligands

`clean_up_option save_best_ligands <value>`

By default, all docked solutions will be kept at the end of a docking run. However, GOLD can produce a lot of output and you may wish to cut this down.

Set `clean_up_option save_best_ligands` in order to retain just the top solution, and for only those `<value>` ligands with the best fitness scores.

Related Instructions:

- `clean_up_option save_fitness_better_than`

clean_up_option delete_redundant_log_files

`clean_up_option delete_redundant_log_files`

By default, a solution log file `<ligand_file_name>_m#.log` is written for each ligand that is docked (see `clean_up_option delete_all_log_files`).

However, under certain circumstances you may wish to use `clean_up_option delete_redundant_log_files` in order to delete unwanted log files, e.g.

- When choosing not to retain all solutions from a docking run (see `clean_up_option save_best_ligands`), you may also wish to remove log files that correspond to solutions which have not been retained, or
- When writing all solutions to a single concatenated file (see `concatenated_output`), then you might not wish to retain log files for individual solutions.

Related Instructions:

- `concatenated_output`
- `clean_up_option save_best_ligands`
- `clean_up_option delete_empty_directories`

clean_up_option save_clustered_solutions

`clean_up_option save_clustered_solutions <value>`

GOLD clusters docked solutions according to how similar the poses are in terms of their RMSD. A link can be generated to the top ranked solution from each distinct cluster. This can be useful in identifying different ligand binding modes.

To generate a link to the top ranked solution from each cluster use the instruction `clean_up_option save_clustered_solutions <value>`, where `<value>` is the RMSD clustering distance (this determines how similar the poses are in each cluster of solutions). By default the clustering distance is 0.75 Å.

A clustering report will be given at the end of the ligand log file. The clusters themselves and the individual solutions within each cluster are listed in ranked order. Symbolic links will also be generated in the output directory which will link to the top-ranked solution in each cluster `cluster_ligand_m#_n.mol2`.

clean_up_option delete_empty_directories

`clean_up_option delete_empty_directories`

When more than one ligand is being docked it is possible to have results for each ligand written to a separate sub-directory (see `make_subdirs`).

However, under certain circumstances you may wish to use `clean_up_option delete_empty_directories` in order to delete any empty output directories, e.g.

- When choosing not to retain all solutions from a docking run (see `clean_up_option save_best_ligands`), you may also wish to remove directories that correspond to solutions which have not been retained, or
- When writing all solutions to a single concatenated file (see `concatenated_output`), then you might not wish to retain empty directories intended for individual solutions.

Related Instructions:

- `make_subdirs`
- `concatenated_output`
- `clean_up_option delete_redundant_log_files`
- `clean_up_option save_best_ligands`

clean_up_option delete_rank_file

`clean_up_option delete_rank_file`

By default, a file called `<ligand_file_name>_m#.rnk` is written for each ligand (`m#` refers to the position of the ligand in the input file). This file contains a summary of the fitness scores for all the docking attempts on that ligand. The docking attempts are listed in decreasing order of fitness score, so the best solution is placed first.

To instruct GOLD not to save ligand `.rnk` files use the instruction `clean_up_option delete_rank_file`

Symbolic links to ranked solutions will also be deleted.

clean_up_option delete_symlinks

`clean_up_option delete_symlinks`

Symbolic links to ligand files may be generated as `cluster_ligand_m#_n.sdf` and `ranked_<ligand_file_name>_m#_n.sdf`, corresponding to the clustering report and ligand rank files, respectively.

To instruct GOLD not to save symbolic links to ranked solutions use the instruction `clean_up_option delete_symlinks`

clean_up_option delete_all_log_files

`clean_up_option delete_all_log_files`

By default, a solution log file `<ligand_file_name>_m#.log` is written for each ligand that is docked (`m#` refers to the position of the ligand in the input file). The log file contains information on: the progress of each docking run, a comparison of the various docking solutions found and information on clustering of ligand poses, for identification of solutions with different binding modes.

To instruct GOLD not to save ligand log files use the instruction `clean_up_option delete_all_log_files`

clean_up_option delete_all_initialised_ligands

`clean_up_option delete_all_initialised_ligands`

By default, each initialised ligand is written to a file with a name of the type `gold_<ligand_filename>_m1.mol2`.

If you do not wish to retain the initialised ligand file for every docked ligand, then specify `clean_up_option delete_all_initialised_ligands` so that each initialised ligand file is deleted at the end of the docking run.

Related Instructions:

- `clean_up_option delete_rank_file`
- `clean_up_option delete_redundant_log_files`

output_file_format

`output_file_format = < MOL2 | MACCS >`

By default, docking solutions will be written out in the same format as was used for input.

To instruct GOLD to write out solution files in an alternative file format use the instruction `output_file_format = < MOL2 | MACCS >`. MOL2 should be used in order to write out files in Tripos MOL2 format and MACCS for MDL SD format.

per_atom_scores

```
per_atom_scores = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

By including the line `per_atom_scores = 1` GOLD will save the scoring contributions of individual ligand and protein atoms to the docked solution output files. For each atom its contribution to the total fitness score and also the constituent scoring terms will be written.

Related Instructions:

- save_per_atom_scores_to_charge_field

save_per_atom_scores_to_charge_field

```
save_per_atom_scores_to_charge_field = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

By including the lines `per_atom_scores = 1` GOLD will save the scoring contributions of individual ligand and protein atoms to the docked solution output files. By also providing the line `save_per_atom_scores_to_charge_field = 1` the ligand atom scores will be saved to the charge field of the solution MOL2 files.

Related instructions:

- per_atom_scores

Write Options

write_options

```
write_options = [NO_LOG_FILES] [NO_LINK_FILES]
[NO_RNK_FILES] [NO_BESTRANKING_LST_FILE]
[NO_GOLD_SOLN_LIGAND_MOL2_FILES] [NO_GOLD_SOLN_LIGAND_SDF_FILES]
[NO_GOLD_LIGAND_MOL2_FILE] [NO_GOLD_PROTEIN_MOL2_FILE]
[NO_LGFNAME_FILE] [NO_PLP_MOL2_FILES] [NO_PID_FILE]
[NO_SEED_LOG_FILE] [NO_GOLD_ERR_FILE] [NO_FIT_PTS_FILES]
[NO_ASP_MOL2_FILES] [NO_GOLD_LOG_FILE] [MIN_OUT]
```

By default, GOLD will produce every output file relevant to your docking. Typically, most, if not all, of these files are valuable to the user. However, in some use cases, such as high-throughput virtual screening, it is not necessary nor desirable for these files to be written to disk. In fact, in some scenarios, it can be costly in both computation and disk space. Hence, it is possible to prevent these files being written using GOLD's write options.

To disable the writing of particular files, include the line `write_options =` followed by one or more of the following keywords:

- `NO_LOG_FILES`: Use this to disable the writing of all ligand log files and the `gold_protein.log` file.
- `NO_LINK_FILES`: Use this to disable the writing of ranked pose shortcut files to solution files. By default, one file is written per solution file.
- `NO_RNK_FILES`: Use this to disable the writing of the ranked fitness lists (`.rnk` extension) for each molecule. By default, one file is written per ligand.
- `NO_BESTRANKING_LST_FILE`: Use this to disable the writing of the `bestranking.lst` file which includes a list of the highest scoring pose for each ligand.
- `NO_GOLD_SOLN_LIGAND_MOL2_FILES`: Use this to disable the writing of all MOL2 solution files. As there would be nothing to point to, this option also disables the writing of the ranked pose shortcut files.
- `NO_GOLD_SOLN_LIGAND_SDF_FILES`: Use this to disable the writing of all SD solution files. As there would be nothing to point to, this option also disables the writing of the ranked pose shortcut files.
- `NO_GOLD_LIGAND_MOL2_FILE`: Use this to disable the writing of all ligand files. By default, one file is written per ligand.
- `NO_GOLD_PROTEIN_MOL2_FILE`: Use this to disable the writing of the protein file. By default, one file is written per target protein.
- `NO_LGFNAME_FILE`: Use this to disable the writing of the `.lgfname` file.

- `NO_PLP_MOL2_FILES`: If using the ChemPLP scoring function, use this to disable the writing of `plp_ligand.mol2` and `plp_protein.mol2`.
- `NO_PID_FILE`: Use this to disable the writing of the `gold.pid` file.
- `NO_SEED_LOG_FILE`: Use this to disable the writing of the `gold.seed_log` file.
- `NO_GOLD_ERR_FILE`: Use this to disable the writing of the `gold.err` file.
- `NO_FIT_PTS_FILES`: Use this to disable the writing of all files related to fitting points including, but not limited to, `fit_pts.mol2` and `fit_pts_merged.mol2`.
- `NO_ASP_MOL2_FILES`: If using the ASP scoring function, use this to disable the writing of `asp_ligand.mol2` and `asp_protein.mol2`.
- `NO_GOLD_LOG_FILE`: Use this to disable the writing of `gold.log`.

Alternatively, you can use the following option to write only the `gold.log` and `bestranking.lst` files. This is the recommended option for high-throughput virtual screening:

- `write_options = MIN_OUT`

Fitness Function Settings

initial_virtual_pt_match_max

```
initial_virtual_pt_match_max = <value>
default: 4.0
```

When GoldScore is being used, the annealing parameters, van der Waals and Hydrogen Bonding, allow poor hydrogen bonds to occur at the beginning of a genetic algorithm run, in the expectation that they will evolve to better solutions.

The parameter `initial_virtual_pt_match_max` is used to set the starting values of max_distance (the distance between donor hydrogen and fitting point must be less than max_distance for the bond to count towards the fitness score). This allows poor hydrogen bonds to occur at the beginning of a GA run.

relative_ligand_energy

```
relative_ligand_energy = <option>
default: 1 (on)
options: 0 (off) | 1 (on)
```

`relative_ligand_energy = 1` is the default setting and which results in the internal energy terms (internal torsion, internal vdw, and internal Hbond) being corrected according to the best energy encountered for these terms during the run. By applying this correction the internal energy will be calculated with respect to that of a close to optimal non-bound structure, thereby taking into account any irreducible internal energy.

gold_fitfunc_path

```
gold_fitfunc_path < goldscore | chemscore | asp | plp |
<filename> | consensus_score >
```

GOLD offers a choice of scoring functions, GoldScore, ChemScore, Astex Statistical Potential, ChemLP and User Defined Score which allows users to modify an existing function.

Scoring functions are implemented in GOLD using shared objects (or dynamically loadable libraries). `gold_fitfunc_path` defines which scoring function is to be used by specifying the path to the relevant dynamically loadable shared object library.

By default the GoldScore scoring function will be used, even if the instruction `gold_fitfunc_path` is not present in the configuration file. To use the ChemScore scoring function it is necessary to include the line:

```
gold_fitfunc_path chemscore
```

To use a modified scoring function set `gold_fitfunc_path` to specify the path to the appropriate shared objects (or dynamically loadable libraries), e.g.

```
gold_fitfunc_path Z:\GOLD\my_score.dll
```

It is possible to perform automatic rescoring on docked poses using a different scoring function to that used during the docking. In order for GOLD to know which scoring functions to use in such a consensus scheme the options `docking_fitfunc_path` and `rescore_fitfunc_path` need to be defined. Further, the `gold_fitfunc_path` option should be set to `consensus_score` and the `run_flag` should be set to `CONSENSUS`. The options `docking_param_file` and `rescore_param_file` are also required to be set when performing automatic rescoring.

Related instructions:

- docking_fitfunc_path
- docking_param_file
- rescore_fitfunc_path
- rescore_param_file
- run_flag

docking_fitfunc_path

```
docking_fitfunc_path < goldscore | chemscore | asp | plp |
        <filename> >
```

It is possible to perform automatic rescoring on docked poses using a different scoring function to that used during the docking. In order for GOLD to know which scoring functions to use in such a consensus scheme the options `docking_fitfunc_path` and `rescore_fitfunc_path` need to be defined. Further, the `gold_fitfunc_path` option should be set to `consensus_score` and the `run_flag` should be set to `CONSENSUS`. The options `docking_param_file`, `rescore_fitfunc_path` and `rescore_param_file` are also required to be set when performing automatic rescoring.

The `docking_fitfunc_path` specifies the scoring function to be used for the docking part of the consensus scoring.

GOLD offers a choice of scoring functions, GoldScore, ChemScore, Astex Statistical Potential, ChemPLP and User Defined Score which allows users to modify an existing function.

Scoring functions are implemented in GOLD using shared objects (or dynamically loadable libraries). `docking_fitfunc_path` defines which scoring function is to be used by specifying the path to the relevant dynamically loadable shared object library.

To use a modified scoring function set `docking_fitfunc_path` to specify the path to the appropriate shared objects (or dynamically loadable libraries), e.g.

```
docking_fitfunc_path Z:\GOLD\my_score.dll
```

Related instructions:

- `gold_fitfunc_path`
- `docking_param_file`
- `rescore_fitfunc_path`
- `rescore_param_file`
- `run_flag`

docking_param_file

```
docking_param_file = < <filename> | DEFAULT >
default: DEFAULT
```

It is possible to perform automatic rescoring on docked poses using a different scoring function to that used during the docking. In order for GOLD to know which scoring function parameter files to use in such a consensus scheme the options `docking_param_file` and `rescore_param_file` need to be defined. Further, the `gold_fitfunc_path` option should be set to `consensus_score` and the

`run_flag` should be set to `CONSENSUS`. The options `docking_fitfunc_path`, `rescore_fitfunc_path` and `rescore_param_file` are also required to be set when performing automatic rescoring.

The `docking_param_file` specifies the scoring function parameter file to be used for the docking part of the consensus scoring.

The scoring function parameter file contains all of the parameters required by the scoring function.

If `docking_param_file = DEFAULT` then the appropriate standard scoring function parameter file provided with the GOLD distribution will be used during the docking run, i.e. either `goldscore.params` or `chemscore.params` will be used depending on whichever scoring function is specified (see `docking_fitfunc_path`).

The scoring function parameter file can be customised by copying it, editing the copy, and instructing GOLD to use the edited file, e.g.

```
docking_param_file = Z:\GOLD\datafiles\my.sf_params
```

The format of the scoring function parameter file is quite strict: incorrect editing may cause GOLD to behave in unexpected ways or even to crash. Because of the large number of parameters, no guarantee can be given that the program will behave reliably with anything other than the default parameterisation.

Specific parameter files for use with heme containing proteins are also available for both GoldScore and ChemScore. For further information, see S. B. Kirton, C. W. Murray, M. L. Verdonk and R. D. Taylor, Proteins: Structure, Function, and Bioinformatics, **58**, 836-844, 2005. The parameters are derived from contact statistics obtained from the CSD and PDB databases. These parameters can be used by specifying the appropriate `.params` file from those that have been supplied with the GOLD installation. The following `.params` files are available within the `$GOLD_DIR/gold` directory:

```
goldscore.p450_csd.params
goldscore.p450_pdb.params
chemscore.p450_csd.params
chemscore.p450.pdb.params
```

A specific parameter file for use with protein kinases is available for ChemScore. For further information, see M. L. Verdonk, V. Berdini, M. J. Hartshorn, W. T. M. Mooij, C. W. Murray, R. D. Taylor, and P. Watson, J. Chem. Inf. Comput. Sci., **44**, 793-806, 2004. This allows weak CHO interactions to be accounted for by inclusion of a ChemScore term that calculates a contribution for weak hydrogen bonds. This term can be useful when dealing with particular proteins, e.g. most kinases contain weak N-heterocycle CH...O hydrogen bonds. The following `.params` file is available within the `$GOLD_DIR/gold` directory:

`chemscore.kinase.params`

Related instructions:

- `gold_fitfunc_path`
- `docking_fitfunc_path`
- `rescore_fitfunc_path`
- `rescore_param_file`
- `run_flag`

rescore_fitfunc_path

```
rescore_fitfunc_path < goldscore | chemscore | asp | plp |
        <filename> >
```

It is possible to perform automatic rescoring on docked poses using a different scoring function to that used during the docking. In order for GOLD to know which scoring functions to use in such a consensus scheme the options `docking_fitfunc_path` and `rescore_fitfunc_path` need to be defined. Further, the `gold_fitfunc_path` option should be set to `consensus_score` and the `run_flag` should be set to `CONSENSUS`. The options `docking_param_file`, `docking_fitfunc_path` and `rescore_param_file` are also required to be set when performing automatic rescoring.

The `rescore_fitfunc_path` specifies the scoring function to be used for the rescoring part of the consensus scoring.

GOLD offers a choice of scoring functions, GoldScore, ChemScore, Astex Statistical Potential, ChemPLP and User Defined Score which allows users to modify an existing function.

Scoring functions are implemented in GOLD using shared objects (or dynamically loadable libraries). `docking_fitfunc_path` defines which scoring function is to be used by specifying the path to the relevant dynamically loadable shared object library.

To use a new or modified scoring function set `rescore_fitfunc_path` to specify the path to the appropriate shared objects (or dynamically loadable libraries), e.g.

```
rescore_fitfunc_path Z:\GOLD\my_score.dll
```

Related instructions:

- `gold_fitfunc_path`
- `docking_fitfunc_path`
- `docking_param_file`
- `rescore_param_file`
- `run_flag`

rescore_param_file

```
rescore_param_file = < <filename> | DEFAULT >
default: DEFAULT
```

It is possible to perform automatic rescoring on docked poses using a different scoring function to that used during the docking. In order for GOLD to know which scoring function parameter files to use in such a consensus scheme the options `docking_param_file` and `rescore_param_file` need to be defined. Further, the `gold_fitfunc_path` option should be set to `consensus_score` and the `run_flag` should be set to `CONSENSUS`. The options `docking_fitfunc_path`, `rescore_fitfunc_path` and `docking_param_file` are also required to be set when performing automatic rescoring.

The `rescore_param_file` specifies the scoring function parameter file to be used for the rescoring part of the consensus scoring.

The scoring function parameter file contains all of the parameters required by the scoring function.

If `rescore_param_file = DEFAULT` then the appropriate standard scoring function parameter file provided with the GOLD distribution will be used during the docking run, i.e. either `goldscore.params` or `chemscore.params` will be used depending on whichever scoring function is specified (see `rescore_fitfunc_path`).

The scoring function parameter file can be customised by copying it, editing the copy, and instructing GOLD to use the edited file, e.g.

```
rescore_param_file = Z:\GOLD\datafiles\my.sf_params
```

The format of the scoring function parameter file is quite strict: incorrect editing may cause GOLD to behave in unexpected ways or even to crash. Because of the large number of parameters, no guarantee can be given that the program will behave reliably with anything other than the default parameterisation.

Specific parameter files for use with heme containing proteins are also available for both GoldScore and ChemScore. For further information, see S. B. Kirton, C. W. Murray, M. L. Verdonk and R. D. Taylor, Proteins: Structure, Function, and Bioinformatics, **58**, 836-844, 2005. The parameters are derived from contact statistics obtained from the CSD and PDB databases. These parameters can be used by specifying the appropriate `.params` file from those that have been supplied with the GOLD installation. The following `.params` files are available within the `$GOLD_DIR/gold` directory:

```
goldscore.p450_csd.params
goldscore.p450_pdb.params
chemscore.p450_csd.params
chemscore.p450.pdb.params
```

A specific parameter file for use with protein kinases is available for ChemScore. For further information, see M. L. Verdonk, V. Berdini, M. J. Hartshorn, W. T. M. Mooij, C. W. Murray, R. D. Taylor, and P. Watson, J. Chem. Inf. Comput. Sci., **44**, 793-806, 2004. This allows weak CHO interactions to be accounted for by inclusion of a

ChemScore term that calculates a contribution for weak hydrogen bonds. This term can be useful when dealing with particular proteins, e.g. most kinases contain weak N-heterocycle CH...O hydrogen bonds. The following `.params` file is available within the `$GOLD_DIR/gold` directory:

```
chemscore.kinase.params
```

Related instructions:

- gold_fitfunc_path
- docking_fitfunc_path
- docking_param_file
- rescore_fitfunc_path
- run_flag

score_param_file

```
score_param_file = < <filename> | DEFAULT >
default: DEFAULT
```

The scoring function parameter file contains all of the parameters required by the scoring function.

If `score_param_file = DEFAULT` then the appropriate standard scoring function parameter file provided with the GOLD distribution will be used during the docking run, i.e. either `goldscore.params` or `chemscore.params` will be used depending on whichever scoring function is specified (see `gold_fitfunc_path`).

The scoring function parameter file can be customised by copying it, editing the copy, and instructing GOLD to use the edited file, e.g.

```
score_param_file = Z:\GOLD\datafiles\my.sf_params
```

The format of the scoring function parameter file is quite strict: incorrect editing may cause GOLD to behave in unexpected ways or even to crash. Because of the large number of parameters, no guarantee can be given that the program will behave reliably with anything other than the default parameterisation.

Specific parameter files for use with heme containing proteins are also available for both GoldScore and ChemScore. For further information, see S. B. Kirton, C. W. Murray, M. L. Verdonk and R. D. Taylor, Proteins: Structure, Function, and Bioinformatics, **58**, 836-844, 2005. The parameters are derived from contact statistics obtained from the CSD and PDB databases. These parameters can be used by specifying the appropriate `.params` file from those that have been supplied with the GOLD installation. The following `.params` files are available within the `$GOLD_DIR/gold` directory:

```
goldscore.p450_csd.params
goldscore.p450_pdb.params
chemscore.p450_csd.params
chemscore.p450.pdb.params
```

A specific parameter file for use with protein kinases is available for ChemScore. For further information, see M. L. Verdonk, V. Berdini, M. J. Hartshorn, W. T. M. Mooij, C. W. Murray, R. D. Taylor, and P. Watson, J. Chem. Inf. Comput. Sci., **44**, 793-806, 2004. This allows weak CHO interactions to be accounted for by inclusion of a ChemScore term that calculates a contribution for weak hydrogen bonds. This term can be useful when dealing with particular proteins, e.g. most kinases contain weak N-heterocycle CH...O hydrogen bonds. The following `.params` file is available within the `$GOLD_DIR/gold` directory:

```
chemscore.kinase.params
```

start_vdw_linear_cutoff

```
start_vdw_linear_cutoff = <value>
default: 3.0
```

When GoldScore is being used, the annealing parameters, van der Waals and Hydrogen Bonding, allow poor hydrogen bonds to occur at the beginning of a genetic algorithm run, in the expectation that they will evolve to better solutions.

At the start of a GOLD run, external van der Waals (vdw) energies are cut off when E_{ij} > van der Waals * k_{ij}, where k_{ij} is the depth of the vdw well between atoms i and j. At the start of the run, the cut-off value is `start_vdw_linear_cutoff`. This allows a few bad bumps to be tolerated at the beginning of the run.

run_flag

```
run_flag = RESCORE [no_simplex] [retrieve] [no_file] |
CONSENSUS [no_simplex]
```

It is possible to rescore a single ligand or a set of ligands in one or more files. Typically, a user will rescore GOLD solution files with an alternative scoring function. However, it is also possible to score a known ligand pose from an alternative source (for example, from a known crystal structure or a solution from another docking program).

To perform a rescoring run include the line `run_flag = RESCORE`. One or more of the following keywords may also be specified:

- `no_simplex`: By default, the docked ligand pose will be minimised before rescoring. Simplexing is important if you are to obtain meaningful scores. Due to the nature of scoring functions, one finds that small changes in location or conformation of the pose can have large effects on the calculated score. To disable simplexing specify the keyword `no_simplex`.
- `retrieve`: When rescoring a GOLD solution file it is possible to use the optimised positions of the polar protein hydrogen atoms that were generated during the original docking. To do this, include the keyword `retrieve`. If this keyword is not used (or no rotatable H positions are available) then the default hydrogen atoms positions specified in the protein input file will be used.

- `no_file`: By default, GOLD will write out docked ligand solutions after rescoring. Solutions will be written to the file `rescore.mol2` (to specify an alternative filename see `concatenated_output`). To disable writing of this file include the keyword `no_file`. If writing of this file is switched off, only the `rescore.log` file will be written.

  Note that rescoring, like docking, requires a fully defined binding site (preferably the same definition that was used for the original docking). The ligand file, scoring function and output preferences must also all be specified.

  It is possible to perform automatic rescoring on docked poses using a different scoring function to that used during the docking. In order for GOLD to know which scoring functions to use in such a consensus scheme the options `docking_fitfunc_path` and `rescore_fitfunc_path` need to be defined. Further, the `gold_fitfunc_path` option should be set to `consensus_score` and the `run_flag` should be set to `CONSENSUS`. The options `docking_param_file` and `rescore_param_file` are also required to be set when performing automatic rescoring.

Related Instructions:

- `concatenated_output`
- `gold_fitfunc_path`
- `docking_fitfunc_path`
- `docking_param_file`
- `rescore_fitfunc_path`
- `rescore_param_file`

alt_residues

```
alt_residues(<1 | 2>) = <residues>
```

GoldScore uses Lennard-Jones functional forms for both the external and internal van der Waals contributions to the fitness function. By default a 6-12 potential is applied to the internal van der Waals contribution and a 4-8 potential is applied to the external

van der Waals contribution. The 4-8 potential form for the external contribution is selected as being optimum for general use. However, there are cases where this potential form may be too severe in the short contact (i.e. the clash) component. This would arise, for instance, where part of the binding site is made up of a loop which it is known can move aside slightly to accommodate large ligands. In such cases it is possible to apply a softer split van der Waals potential for certain selected residues. Two alternative soft split potential forms are parameterised in the `gold.params` file:

```
EXTERNAL_POTENTIAL(1) = 4-8 2-4 - Form 1
EXTERNAL_POTENTIAL(2) = 4-8 1-2 - Form 2
```

The first term of each form describes long range interactions, the second term describes short range interactions. One of these two soft potentials can be applied to a single residue using the instruction:

```
alt_residues(form) = <residue>
```

Where form is the Split Potential form to be applied (i.e. 1 or 2), and `<residue>` is the residue to which the split potential is to be applied, e.g. specifying

```
alt_residues(1) = ALA148
```

will apply the split potential of form 1 to the residue ALA148. More than one residue can be specified, and both potential forms can be used in the same GOLD run.

Protein Data

protein_datafile

```
protein_datafile = <filename>
```

`protein_datafile` is used to provide GOLD with a file containing the protein (or the part of a protein) into which the ligand is to be docked, `protein_datafile` must be followed by a filename, e.g.
`protein_datafile = Z:\GOLD\datafiles\protein.mol2`

Acceptable protein file formats are PDB and MOL2. Before being used with GOLD the protein input file must have been prepared in accordance with the guidelines provided, using a good modelling package.

Receptor Depth Scaling

receptor_depth_scaling

```
receptor_depth_scaling = <option>
default: 0 (off)
options: 0 (off) | 1 (on)
```

Receptor depth scaling (RDS) can be chosen when performing a rescore using ChemScore as the scoring function. RDS will reward hydrogen bonds deep in protein pockets with an increased score, while the scores of those closer to the solvent-exposed surface are decreased. Simultaneously, the scores attributed to lipophilic interactions are reduced.

To enable the use of RDS in GOLD set `receptor_depth_scaling = 1`, all RDS required instructions need to be set in order for RDS to work properly

Required instructions:

- `rds_use_protein_coords`
- `rds_use_donor_coords`
- `rds_use_exact_count`
- `rds_protein_distance`
- `rds_hbond`
- `rds_lipo`
- `rds_clash`
- `rds_metal`

rds_use_protein_coords

```
rds_use_protein_coords = <option>
default: 1(on)
options: 0 (off) | 1 (on)
```

Use the protein atom position (coordinates) when calculating the depth of the atom involved in the interaction. Set `rds_use_protein_coords = 0` to use the ligand atom position instead.

Required instructions:

- receptor_depth_scaling
- rds_use_donor_coords
- rds_use_exact_count
- rds_protein_distance
- rds_hbond
- rds_lipo
- rds_clash
- rds_metal

rds_use_donor_coords

```
rds_use_donor_coords = <option>
default: 1(on)
options: 0 (off) | 1 (on)
```

When the receptor depth is calculated for a hydrogen bond donor the heavy atom position (typically, N or O) is used per default. Set to `0` (off) to use the position of the hydrogen for the calculation.

Required instructions:

- receptor_depth_scaling
- rds_use_protein_coords

- rds_use_exact_count
- rds_protein_distance
- rds_hbond
- rds_lipo
- rds_clash
- rds_metal

rds_use_exact_count

```
rds_use_exact_count = <option>
default: 1(on)
options: 0 (off) | 1 (on)
```

Receptor depth (RD) values need to be pre-calculated on a grid to speed up the rescore when ligand atoms are involved. For hydrogen bond interactions where `rds_use_protein_coords = 1` the exact values of the RD may be pre-calculated for the protein atom positions and used instead.

Note: Lipophilic interactions will still require the grid-based count.

Note: Only possible if `rds_use_protein_coords = 1` (see rds_use_protein_coords)

Required instructions:

- receptor_depth_scaling
- rds_use_protein_coords
- rds_use_donor_coords
- rds_protein_distance
- rds_hbond
- rds_lipo
- rds_clash

- rds_metal

rds_protein_distance

```
rds_protein_distance = <value>
default: 8
```

This is the radius (Å) of the sphere used to calculate the receptor depth. Atoms within this radius are counted in order to evaluate the depth of the interaction.

Required instructions:

- receptor_depth_scaling
- rds_use_protein_coords
- rds_use_donor_coords
- rds_use_exact_count
- rds_hbond
- rds_lipo
- rds_clash
- rds_metal

rds_hbond

```
rds_hbond = <min. no. atoms> <max. no. atoms> <scale factor
1> <scale factor 2> <output>

default: rds_hbond = 13 105 0 1.8 0
```

This parameter controls the scaling of the hydrogen bond interactions in the RDS calculation. If the number of atoms surrounding a hydrogen bond between the ligand and the protein is <min. no. atoms> or less it is scaled by <scale factor 1>. Between <min. no. atoms> and <max. no. atoms> it is scaled linearly to <scale factor 2>. All hydrogen bonds with more than <max. no. atoms> are

scaled by <scale factor 2>. The variable <output> is 0 for default output while 1 is for verbose output. Note: Output set to 1 will give large amounts of information.

The receptor depth scaling has been validated for the default values only, any change of these values can lead to unpredictable results and should be done with caution.

Required instructions:

- receptor_depth_scaling
- rds_use_protein_coords
- rds_use_donor_coords
- rds_use_exact_count
- rds_protein_distance
- rds_lipo
- rds_clash
- rds_metal

rds_lipo

```
rds_lipo = <min. no. atoms> <max. no. atoms> <scale factor
1> <scale factor 2> <output>
```

```
default: rds_lipo = 0 0 0.52 0.52 0
```

The rds_lipo setting controls how the lipophilic interactions are scaled within RDS. If the number of atoms surrounding an interaction point between the ligand and the protein is <min. no. atoms> or less it is scaled by <scale factor 1>. Between <min. no. atoms> and <max. no. atoms> it is scaled linearly to <scale factor 2>. All interactions with more than <max. no. atoms> are scaled by <scale factor 2>. The variable <output> is 0 for default output while 1 is for verbose output. Note: Output set to 1 will give large amounts of information.

The receptor depth scaling has been validated for the default values only, any changes of these values can lead to unpredictable results and should be done with caution. The default settings, i.e. `<min. no. atoms> = 0` and `<max. no. atoms> = 0`, entail that all lipophilic interactions within the RDS are scaled by 0.8.

Required instructions:

- receptor_depth_scaling
- rds_use_protein_coords
- rds_use_donor_coords
- rds_use_exact_count
- rds_protein_distance
- rds_hbond
- rds_clash
- rds_metal

rds_clash

```
rds_clash = <min. no. atoms> <max. no. atoms> <scale factor
1> <scale factor 2> <output>

default: rds_clash = 0 0 1 1 0
```

The `rds_clash` setting controls how the clash term is scaled within RDS. If the number of atoms surrounding an interaction point between the ligand and the protein is `<min. no. atoms>` or less it is scaled by `<scale factor 1>`. Between `<min. no. atoms>` and `<max. no. atoms>` it is scaled linearly to `<scale factor 2>`. All interactions with more than `<max. no. atoms>` are scaled by `<scale factor 2>`. The variable `<output>` is 0 for default output while 1 is for verbose output. Note: Output set to 1 will give large amounts of information.

The receptor depth scaling has been validated for the default values only, any changes of these values can lead to unpredictable results and should be done with caution. The default settings, i.e. `<min. no. atoms> = 0` and `<max. no. atoms> = 0`, entail that the clash term within the RDS is scaled by 1.

Required instructions:

- receptor_depth_scaling
- rds_use_protein_coords
- rds_use_donor_coords
- rds_use_exact_count
- rds_protein_distance
- rds_hbond
- rds_lipo
- rds_metal

rds_metal

```
rds_metal = <min. no. atoms> <max. no. atoms> <scale factor
1> <scale factor 2> <output>

default: rds_metal = 13 105 0 1.8 0
```

The `rds_metal` setting controls how the metal interaction term is scaled within RDS. If the number of atoms surrounding an interaction point between the ligand and the protein is `<min. no. atoms>` or less it is scaled by `<scale factor 1>`. Between `<min. no. atoms>` and `<max. no. atoms>` it is scaled linearly to `<scale factor 2>`. All interactions with more than `<max. no. atoms>` are scaled by `<scale factor 2>`. The variable `<output>` is 0 for default output while 1 is for verbose output. Note: Output set to 1 will give large amounts of information.

The receptor depth scaling has been validated for the default values only, any changes of these values can lead to unpredictable results and should be done with caution.

Required instructions:

- `receptor_depth_scaling`
- `rds_use_protein_coords`
- `rds_use_donor_coords`
- `rds_use_exact_count`
- `rds_protein_distance`
- `rds_hbond`
- `rds_lipo`
- `rds_clash`

Water Data

water

```
water <atom id> < on | off | toggle > < spin | fix |
trans_spin <translation_value> > <full_water_file_path>
```

GOLD allows waters to switch on and off (i.e. to be bound or displaced) and to rotate around their three principal axes (to optimise hydrogen bonding) during docking.

Setting the `trans_spin` option will make GOLD spin and translate the water molecule to optimise the orientation of the hydrogen atoms as well as the water molecule's position within a user defined radius (corresponding to `translation_value`). Note that the translation value must be between 0 and 2 Å, e.g. `water 1 toggle trans_spin 1.5`.

To predict whether a specific water molecule should be bound or displaced, GOLD estimates the free-energy change, ΔG_b, associated with transferring a water molecule from the bulk solvent to its binding site in a protein-ligand complex. Further details can be found in Modeling Water Molecules in Protein-Ligand Docking Using GOLD. Marcel L. Verdonk, Gianni Chessari, Jason C. Cole, Michael J. Hartshorn, Christopher W. Murray, J. Willem M. Nissink, Richard D. Taylor, and Robin Taylor, J. Med. Chem., **48**, 6504-6515, 2005.

For each key water molecule, you will need to specify:

- The atom number of the water oxygen atom (as defined in the protein input MOL2 file).
- The state of the water, available options are:
  - `on`: use the water for docking (i.e. present).
  - `off`: do not use the water for docking (i.e. absent).
  - `toggle`: have GOLD decide whether the water should be present or absent (i.e. bound or displaced) during the docking run.
- The orientation of the water hydrogen atoms, available options are:
  - `fix`: use the orientation specified in the input file.
  - `spin`: have GOLD automatically optimise the orientation of the hydrogen atoms.
  - `trans_spin`: have GOLD optimise the orientation of the water H atoms, as well as optimise the location of the water O atom within a user-defined radius (`translation_value`) of less than 2 Å.
- The full directory path (`full_water_file_path`) where the water file is stored in (because waters are able to move and are specified outside the protein they need their own file path).

An example of acceptable input is

```
water 1 toggle trans_spin 2 /home/gold/waters/H2O126.mol2
```

Any unspecified waters that are part of the protein are considered to be `on` automatically and their orientation will not be optimised during docking.

Metal Data

metal_coordination_spec

```
metal_coordination_spec
point <value> <value> <value>
point <value> <value> <value>
point <value> <value> <value>
....
end_metal_coordination_spec
```

By default, GOLD will automatically determine metal coordination geometries. The following geometries are recognised:

Template	Geometry	Coordination Number
TETR	Tetrahedral	n=4
TBP	Trigonal bipyramidal	n=5
OCT	Octahedral	n=6
CTP	Capped trigonal prism	n=7
PBP	Pentagonal bipyramidal	n=7
SQAP	Square prism	n=8
ICO	Icosahedral	n=10
DOD	Dodecahedral	n=12

In order to determine the coordination geometry of a particular metal atom GOLD performs a permuted superimposition of coordination geometry templates onto the coordinating atoms found in the protein. Coordination fitting points are then generated using the template that gives the best fit (based on RMSD).

The geometry templates used for given metals are defined in the `gold.params` file in the section headed `# Metals`. For example, for a Zn atom GOLD will attempt to match coordination geometries 4, 5 and 6 (tetrahedral, trigonal bipyramidal, and octahedral templates) onto the coordinating atoms found in the protein. The template that gives the best match will then be used to generate coordination fitting points.

In addition to the templates listed above it is possible to specify custom metal coordination geometries which can subsequently be used to derive ligand binding points around particular metal atoms.

Custom metal polyhedron may contain up to nine points. Each point in the custom polyhedron must be specified using a vector (assuming the centre of your polyhedron is at the origin).

For example, to set up a custom square planar geometry you must specify four points using the following instructions:

```
metal_coordination_spec
point 0 1 0
point 1 0 0
point -1 0 0
point 0 -1 0
end_metal_coordination_spec
```

Assuming the metal is on the origin (0,0,0), GOLD will then attempt to match the specified vectors onto the metal-to-protein-atom vectors found in the protein (vectors are normalised to a metal-to-chelator distance of 2.0 Å).

Once defined, it is necessary to explicitly instruct GOLD to consider custom metal coordination geometries when matching templates onto the coordinating atoms found in the protein (see `overrule_metal_coordination`).

Related Instructions:

- overrule_metal_coordination

overrule_metal_coordination

`overrule_metal_coordination <atom id> <coordination geometries>`

By default, GOLD will automatically determine metal coordination geometries.

In order to determine the coordination geometry of a particular metal atom GOLD performs a permuted superimposition of coordination geometry templates (tetrahedral, octahedral, etc.) onto the coordinating atoms found in the protein. Coordination fitting points are then generated using the template that gives the best fit (based on RMSD).

The geometry templates used for given metals are defined in the `gold.params` file in the section headed `# Metals`. For example, for a Zn atom GOLD will attempt to match coordination geometries 4, 5 and 6 (tetrahedral, trigonal bipyramidal, and octahedral templates) onto the coordinating atoms found in the protein. The template that gives the best match will then be used to generate coordination fitting points.

It is possible to manually specify coordination geometries for particular metal atoms. This can be used to allow non-standard metal coordination geometries, or to limit the number of possible geometries that GOLD checks, i.e. it is possible to overrule the default geometries for the corresponding metal type defined in the `gold.params` file.

The atom number of the metal (as defined in the protein input file) must be specified. This should be followed by a comma-separated list of the allowed coordination numbers. Only the templates that correspond to these specified coordination numbers will be used for matching.

For example, the following instruction:

```
overrule_metal_coordination 4049 4,6
```

will specify that metal atom 4049, a Zn atom, must be matched against tetrahedral (4) and octahedral (6) coordination geometries only, i.e. excluding 5 (trigonal bipyramid).

To specify a custom or non-standard metal coordination geometry (see `metal_coordination_spec`) you must use a negative coordination number.

For example, the following instruction:

```
overrule_metal_coordination 4049 4,-4
```

will specify that metal atom 4049 must be matched against tetrahedral (4) and a custom square planar (-4) geometry only.

Related Instructions:

- metal_coordination_spec

Protein Data

ensemble_structure

```
ensemble_structure
protein_datafile = <filename>
protein_score_offset = <value>
end_ensemble_structure
```

It is possible to dock ligands into multiple proteins in one docking run, i.e. to carry out an ensemble docking. Starting from a superimposed set of protein structures, GOLD evolves a separate population of individuals (representing ligand conformations) for each protein structure that is part of the ensemble. The best ligand conformation found in any of the ensemble structures is returned.

For each protein involved in the ensemble it is necessary to add an `ensemble_structure` block of commands to the gold.conf. Within this block a number of parameters can be defined such as the protein filename, any score offset values, any constraints, flexible sidechains or customised metal geometries, e.g.

```
ensemble_structure
protein_datafile = 1QPD_protein.mol2
protein_score_offset = 5.0
```

```
CONSTRAINTS
constraint protein_h_bond 10.0000 0.005000 1207
end_ensemble_structure
```

The `protein_datafile` instruction specifies the input protein file.

The `protein_score_offset` instruction must be accompanied by a value which will be subtracted from the overall fitness score if a ligand is docked into this protein structure. In this way the selection of certain protein structures can be biased. If using this feature, these scores are reported as `DE (Protein)` in the GOLD log files.

Constraints that are specific to the protein e.g. a protein H bond constraint should be specified within the `ensemble_structure` block as above. Similarly with flexible sidechains and customised metal geometries. Instructions for specifying constraints, flexible sidechains and customised metal geometries are detailed elsewhere in this document.

Related Instructions:

- `constraint_distance`
- `constraint_h_bond`
- `constraint_protein_h_bond`
- `constraint_substructure`
- `protein_datafile`
- `rotamer_lib`

Flexible Sidechains

rotamer_lib

```
rotamer_lib
name <identifier>
chi<value> <value> <value> <value> <value>
```

```
rotamer <value|value (value)|value (value:value)>
end_rotamer_lib
```

You may specify that one or more protein sidechains are to be treated as flexible. Each flexible sidechain will be allowed to undergo torsional rotation around one or more of its acyclic bonds during docking.

For each sidechain that you want to make flexible, you should add a `rotamer_lib` block of commands to the `gold.conf` file. This specifies the name of the sidechain, the torsion angles that are permitted to vary, and the allowed values or ranges of values for those torsion angles. You can have up to 10 `rotamer_lib` blocks in a given configuration file, each one pertaining to a particular protein sidechain. For example, consider the following `rotamer_lib` command block:

```
rotamer_lib
name tyr370
chi1 497 498 501 502
chi2 498 501 502 503
rotamer 60 90
rotamer -60 (10) -85 (10:15)
end_rotamer_lib
```

`name` is used to specify a unique identifier for the `rotamer_lib` command block. Any text can be used but the obvious choice is the name of the side chain that the command block refers to, in this case `tyr370`.

The `chi1` command specifies the atom numbers of the atoms defining the first rotatable torsion. The atom indices as they appear in the input file must be specified. In the example, this corresponds to rotation around Cα-Cβ, so the atoms will be the backbone N (= atom 497), CA (498), CB (501) and CG (502). It is necessary to specify the atoms from the backbone outwards, i.e. `chi1 502 501 498 497` would be invalid.

The `chi2` command specifies the second rotatable torsion. In this example, this corresponds to rotation around Cβ-Cγ, so the atoms are CA (498), CB (501), CG (502) and CD1 (503).

You may specify up to 8 `chi` commands in a given `rotamer_lib` block.

Each `rotamer` command describes one allowed conformation for the sidechain. Thus, the first `rotamer` command specifies the first set of allowed values for `chi1` and `chi2`. In the example, this is `chi1 = 60, chi2 = 90`.

The second `rotamer` command specifies the second set of allowed values. The format x (y) specifies the range (x - y) to (x + y), while x (y:z) specifies the range (x - y) to (x + z). In the example, therefore, chi1 is allowed to have any value between -70 and -50 degrees, and chi2 is allowed to vary continuously between -95 and -70 degrees.

In summary, the effect of this `rotamer_lib` command block is therefore to allow Tyr370 to adopt the conformation chi1 = 60, chi2 = 90, or any conformation in the range chi1 = -70 to -50, chi2 = -95 to -70.

You can have up to 50 `rotamer` commands in a `rotamer_lib` block.

Quite often, a sidechain rotation is accompanied by a small change in the local backbone conformation, primarily affecting the position of the Cγ atom. Although minor, this movement is extremely important because it alters the vector direction Cα-Cβ, and this can have a big leverage effect on the positions of atoms further down the sidechain. The backbone movement can be mimicked by allowing the Cα atom and the attached sidechain to rotate around the N-C vector, where N and C are the backbone atoms on either side of the Cα atom. This is defined as a rotation of the improper torsion defined by the atom sequence CA-N-C-CA.

The file `<GOLD_DIR>/gold/rotamer_library.txt` contains information taken from the paper The Penultimate Rotamer Library, S. C. Lovell, J. M. Word, J. S. Richardson & D. C. Richardson, Proteins, **40**, 389-408, 2000. This is a compilation of the most commonly observed sidechain conformations for the naturally occurring amino acids. To make use of the rotamer information for a given residue, copy and paste the relevant `rotamer_lib` section into the GOLD configuration file and specify the residue name and atom numbers as required. Note that the library settings are simply a starting point; users are encouraged to generate their own rotamers for optimal results.

Related Instructions:

- energy
- per_atom_scores

energy

```
energy <value>
```

An energy may be assigned to a given rotamer, e.g. as follows:

```
rotamer_lib
    name tyr370
    chi1 497 498 501 502
    chi2 498 501 502 503
    rotamer 62 (11) 90 (11)
    energy 10
    rotamer -65 (11) -85 (18)
    end_rotamer_lib
```

This will penalise (i.e. reduce) the fitness score value by 10 units if the Tyr370 side chain is placed in the chi1 = 62, chi2 = 90 conformation. In other words, it makes this conformation less favourable.

Had the command `energy -10` been included, its effect would have been to improve (i.e. increase) the fitness score value.

Related Instructions:

- rotamer_lib

penalise_protein_clashes

```
penalise_protein_clashes = <option>

default: 1 (on)
options: 0 (off) | 1 (on)
```

By default, when a flexible sidechain is moved during docking (see `rotamer_lib`), GOLD checks whether any of its atoms clash with atoms in neighbouring residues. This gives rise to an extra Protein Energy term which contributes to the total fitness score value.

The term is computed by summing the van der Waals interactions of all pairs of protein atoms which satisfy the following conditions: (a) at least one of the protein atoms is in a flexible side chain; (b) the van der Waals term for that pair of atoms is repulsive. The van der Waals interactions will be estimated using the same potential as is used for the protein-ligand vdw term.

Setting `penalise_protein_clashes = 0` will switch off calculation of the protein-protein clash term for all flexible side chains, not just the one corresponding to the `rotamer_lib` block in which you have placed the `penalise_protein_clashes = 0` command.

Related Instructions:

- `rotamer_lib`

Internal

seed_file

`seed_file = <filename>`

It is possible to supply the seed for the random number generator used by the genetic algorithm in GOLD.

By default, the file `gold.seed_log` will be generated automatically during a GOLD run. This file will be written to your specified output directory. However, it is possible to instruct GOLD to use a `gold.seed_log` file from a previous calculation, or to use a customised `gold.seed_log` file.

`seed_file` is used to provide GOLD with the location of the `gold.seed_log` file.

This provides a mechanism for introducing a non-random start facility. Normally, a new GOLD calculation will be seeded in this way in order to reproduce identical results for repeat runs. You could also use this mechanism to manually specify a seed for a new GOLD calculation.