RxDock Documentation

Release 0.1.0

RxDock development team

Mar 20, 2020

CONTENTS

RxDock is a fast and versatile **open-source docking program** that can be used to dock **small molecules** against **proteins** and **nucleic acids**. It is designed for high-throughput virtual screening (HTVS) campaigns and binding mode prediction studies.

RxDock is mainly written in C++ and accessory scripts and programs are written in C++, Perl or Python languages.

The full RxDock software package requires **less than 50 MB** of hard disk space and it is compilable (at this moment, **only**) in **all Linux computers**.

Thanks to its design and implementation [rDock2014], it can be installed on a computation cluster and deployed on an **unlimited number of CPUs**, allowing HTVS campaigns to be carried out in a **matter of days**.

Besides its main Docking program, the RxDock software package also provides a set of tools and scripts to facilitate **preparation** of the input files and **post-processing** and **analysis** of results.

CHAPTER

ONE

ABOUT

1.1 Download

Please visit RxDock GitLab page for most up to date releases.

- Download a released version
- Get the latest code using git

1.2 Features

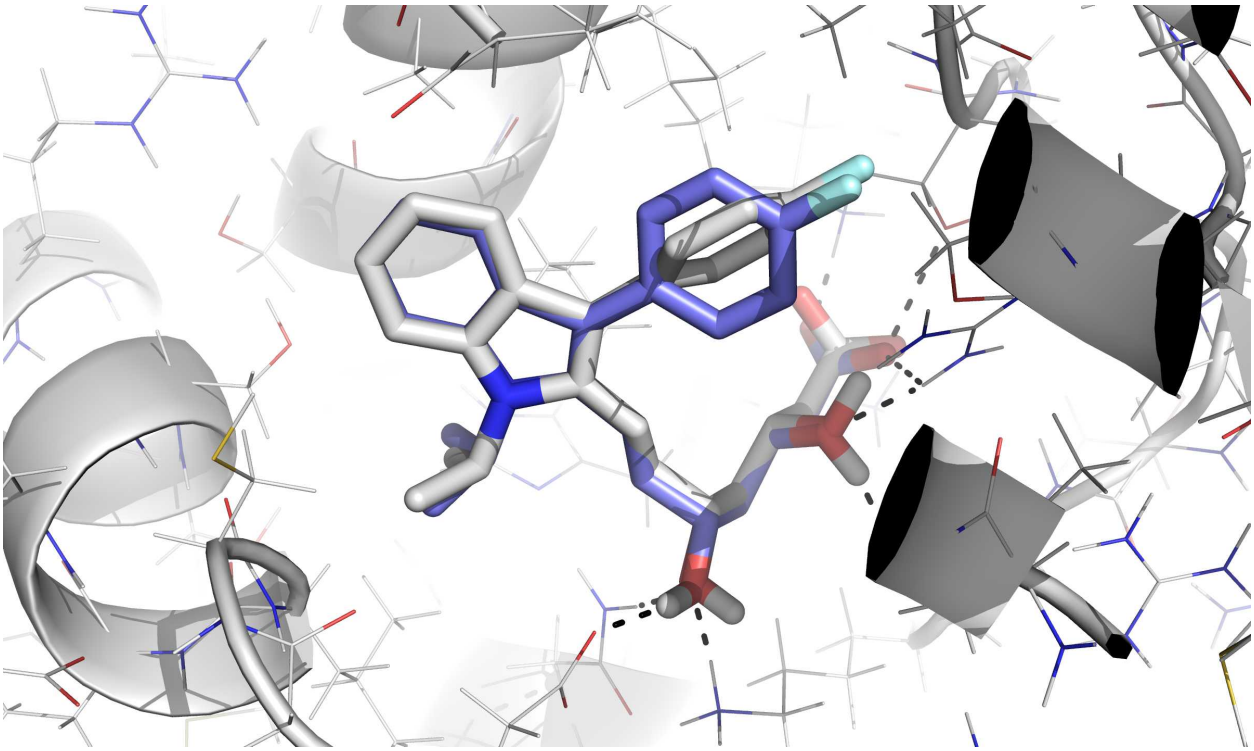


Fig. 1.1: The above image illustrates the first binding mode solution for ASTEX system 1hwi, with an RMSD of 0.88 Å.

Docking preparation Define cavities using **known binders** or with user-supplied **3D coordinates**. Allow -OH and -NH2 receptor side chains to rotate. Add explicit solvent molecules and structural waters. Supply pharmacophoric restraints as a bias to **guide docking**.

Pre-processing of input files Define common ligand structure for performing **tethered docking** (requires Open Babel Python bindings). Sort, filter or split ligand files for facilitating **parallelization**. Find **HTVS protocol** for optimizing calculation time. Pre-calculate grids to decrease subsequent calculation times.

Post-processing and analysis of results Summarize results in a tabular format. Sort, filter, merge or split results files. Calculate **RMSD** with a reference structure taking into account internal symmetries (requires Open Babel Python bindings).

Binding mode prediction Predict how a ligand will bind to a given molecule. The ASTEX non-redundant test set for proteins and DOCK and RxDock test sets for RNA have been used for validating and comparing RxDock with other programs.

High-throughput virtual screening Run for million of compounds in short time by exploiting the capabilities of computer calculation farms. Ease of **parallelization** in relatively unlimited CPUs to optimize HTVS running times. The DUD set has been used for validating RxDock and comparing its performance to other reference docking programs.

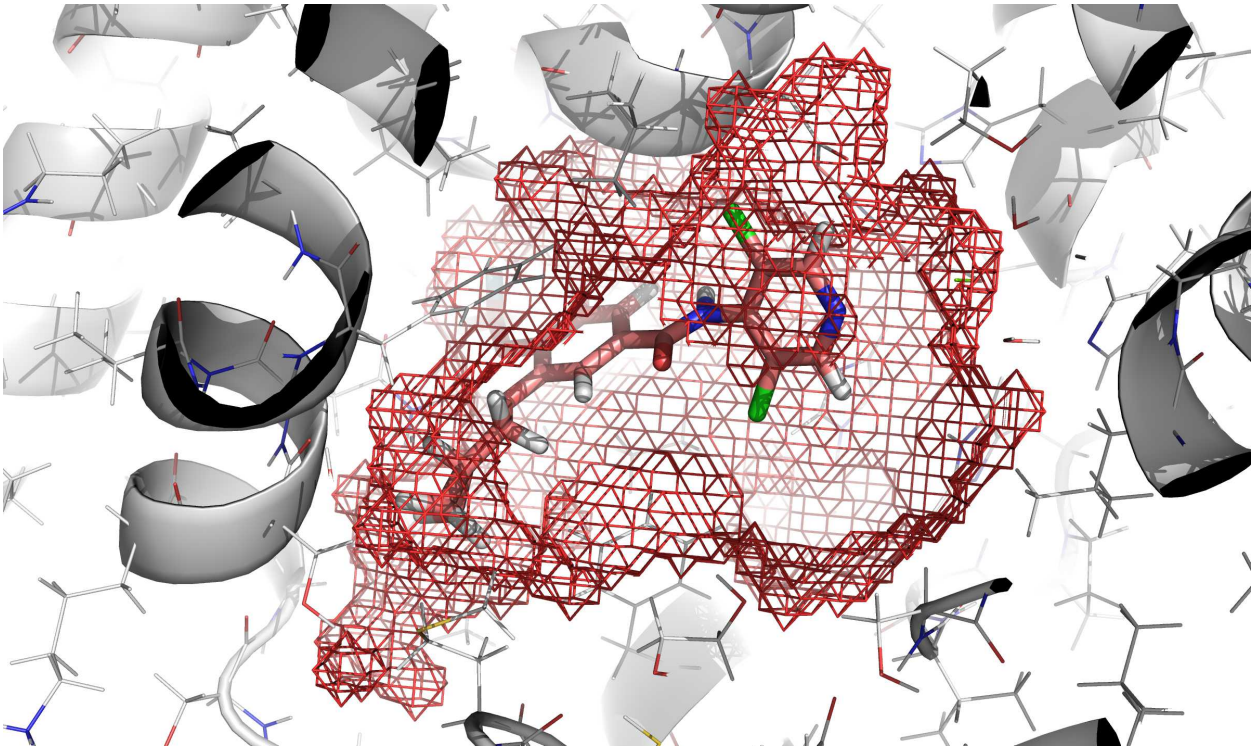


Fig. 1.2: In red mesh, definition of the cavity obtained by execution of `rbcavity` program.

1.3 History

The RiboDock program was developed from 1998 to 2006 by the software team at RiboTargets (subsequently Vernalis (R&D) Ltd) [RiboDock2004]. In 2006, the software was licensed to the University of York for maintenance and distribution under the name rDock.

In 2012, Vernalis and the University of York agreed to release the program as open-source software [rDock2014]. This version is developed with support from the University of Barcelona – sourceforge.net/projects/rdock.

The development of rDock stalled in 2014. Since 2019, RxTx is developing a fork of rDock under the name RxDock.

1.4 License

RxDock is licensed under GNU LGPL version 3.0.

1.4.1 GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

1.5 Contributor concordat

RxDock adheres both to No Code of Conduct and Code of Merit.

1.5.1 Contributor Code of Conduct

This project adheres to No Code of Conduct. We are all adults. We accept anyone's contributions. Nothing else matters.

For more information please visit the No Code of Conduct homepage.

1.5.2 Code of Merit

1. The project creators, lead developers, core team, constitute the managing members of the project and have final say in every decision of the project, technical or otherwise, including overruling previous decisions. There are no limitations to this decisional power.
2. Contributions are an expected result of your membership on the project. Don't expect others to do your work or help you with your work forever.
3. All members have the same opportunities to seek any challenge they want within the project.
4. Authority or position in the project will be proportional to the accrued contribution. Seniority must be earned.
5. Software is evolutive: the better implementations must supersede lesser implementations. Technical advantage is the primary evaluation metric.
6. This is a space for technical prowess; topics outside of the project will not be tolerated.
7. Non technical conflicts will be discussed in a separate space. Disruption of the project will not be allowed.
8. Individual characteristics, including but not limited to, body, sex, sexual preference, race, language, religion, nationality, or political preferences are irrelevant in the scope of the project and will not be taken into account concerning your value or that of your contribution to the project.
9. Discuss or debate the idea, not the person.
10. There is no room for ambiguity: Ambiguity will be met with questioning; further ambiguity will be met with silence. It is the responsibility of the originator to provide requested context.
11. If something is illegal outside the scope of the project, it is illegal in the scope of the project. This Code of Merit does not take precedence over governing law.
12. This Code of Merit governs the technical procedures of the project not the activities outside of it.
13. Participation on the project equates to agreement of this Code of Merit.
14. No objectives beyond the stated objectives of this project are relevant to the project. Any intent to deviate the project from its original purpose of existence will constitute grounds for remedial action which may include expulsion from the project.

This document is the Code of Merit, version 1.0.

1.6 References

If you are using RxDock in your research, please cite [rDock2014]. Former software reference provided for completeness is [RiboDock2004].

CHAPTER

TWO

GETTING STARTED GUIDE

In this section, you have the documentation with installation and validation instructions for first-time users.

To continue with a short validation experiment (contained in the *Getting started guide*), please visit the *Validation experiments section*.

2.1 Overview

RxDock is a fast and versatile open-source docking program that can be used against proteins and nucleic acids. It is designed for high-throughput virtual screening (HTVS) campaigns and binding mode prediction studies.

The RxDock program was developed from 1998 to 2006 (formerly known as RiboDock [RiboDock2004]) by the software team at RiboTargets (subsequently Vernalis (R&D) Ltd.). In 2006, the software was licensed to the University of York for maintenance and distribution under the name rDock. In 2012, Vernalis and the University of York agreed to release the program as open-source software. The released version is licensed under GNU LPGL version 3.0 with support from the University of Barcelona – rdock.sourceforge.net. The development of the open-source code stopped in 2014, so in 2019 RxTx decided to revive it by forking rDock as RxDock.

The major components of the platform now include fast intermolecular scoring functions (van der Waals, polar, desolvation) validated against protein and RNA targets, a Genetic Algorithm-based stochastic search engine, a wide variety of external structure-based drug discovery (SBDD) derived restraint terms (tethered template, pharmacophore, noe distance restraints), and novel Genetic Programming-based post-docking filtering. A variety of scripts are provided to perform automated validation experiments and to launch virtual screening campaigns.

This introductory guide is aimed at new users of RxDock. It describes the minimal set of steps required to build RxDock from the source code distribution, and to run one of the automated validation experiments provided in the test suite distribution. The instructions assume that you are comfortable with simple Linux command line administration tasks, and with building Linux application from makefiles. Once you are familiar with these steps you should proceed to the User and Reference Guide for more detailed documentation on the usage of RxDock.

2.2 Quick and dirty installation

In this section you will have short instructions to make a typical installation of RxDock.

To get the full documentation of all RxDock software package and methods, please go to the *Reference guide*.

Moreover, you can also check the following information:

- *Getting started*: installation and validation instructions for first-time users.
- *Validation experiments*: instructions and examples for re-running the validation sets we have carried out.

- *Calculating ROC curves*: tutorial for generating ROC curves and other statistics after running RxDock docking jobs.

2.2.1 Installation in 3 steps

We have been able to compile RxDock in the following Linux systems:

- CentOS 5.5 64 bits
- openSUSE 11.3 32 and 64 bits
- openSUSE 12.3 32 and 64 bits
- openSUSE 13.1 32 and 64 bits
- Ubuntu 12.04 32 and 64 bits

Step 1

First of all, you will need to install several packages before compiling and running RxDock:

- gcc and g++ compilers version > 3.3
- make
- cppunit and cppunit-devel

Note: For Ubuntu users:

If you are trying to use RxDock in Ubuntu, please note that csh shell is not included in a default installation. We recommend to install csh in case some error arises (`sudo apt-get install csh`), even with all the above-stated dependencies installed.

Afterwards, download the source code compressed file or get it by SVN in *Downloads* section.

Step 2

Then, run the following commands:

```
$ tar -xvzf rxdock-0.1.0.tar.gz
$ cd rxdock-0.1.0/build/
```

and, for 32 bits computers:

```
$ make linux-g++
```

for 64 bits computers:

```
$ make linux-g++-64
```

Step 3

After compiling successfully, type the following command to make a test and check that your compiled version works good and the results are correct.

```
$ make test
```

If the test has succeed, you are done, enjoy using RxDock!

Otherwise, please check your dependencies and all the previous commands or go to *Support Section* to ask for help.

Just as a concluding remark, don't forget to set the necessary environmental variables for running RxDock in the command line (for example, in Bash shell):

```
$ export RBT_ROOT=/path/to/rxdock/installation/
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$RBT_ROOT/lib
$ export PATH=$PATH:$RBT_ROOT/bin
```

2.3 Prerequisites

2.3.1 Compilers

RxDock is supplied as source code, which means that you will have to compile the binary files (run-time libraries and executable programs) before you use them. RxDock has been developed largely on the Linux operating systems, most recently with GNU g++ compiler (tested under openSUSE 11.3). The code will almost certainly compile and run under other Linux distributions with little or no modification. For the moment, it has been tested in the latest Ubuntu and openSUSE releases for both 32 and 64 bits system architectures (by November 2013) and compilation was possible without any code modification. However, no other distributions or compilers have been tested extensively to date.

For full production use, you would typically compile RxDock on a separate build machine and run the docking calculations on a cluster of compute machines. However, for the purposes of getting started, these instructions assume that you will be compiling RxDock and running the initial validation experiments on the same machine.

2.3.2 Required packages

Make sure you have the following packages installed on your machine before you continue. The versions listed are appropriate for openSUSE 11.3; other versions may be required for your particular Linux distribution.

Table 2.1: Required packages for building and running RxDock

Package	Description	Required at	Version
`gcc`	GNU C compiler	Compile-time	>=3.3.4
`g++`	GNU C++ compiler	Compile-time	>=3.3.4
`gtest`	Google's C++ unit testing framework	Compile-time	>=1.8.0
`gtest-devel`	Development files for gtest	Compile-time	>=1.8.0

2.4 Unpacking the distribution files

The RxDock source files and test suite files are provided as independent gzipped tar (.tar.gz) distributions. Depending on your requirements, the two distributions can be unpacked to entirely separate locations, or can be unpacked under the same location. In this example they are unpacked under the same location.

Table 2.2: RxDock distribution files

File	Description
`rxdock-[CODELINE].tar.gz`	RxDock source distribution
`[TEST]_rDock_TestSet.tar.gz`	Test suite data files and scripts

Here `[CODELINE]`, and `[TEST]` will vary depending on the release and test set. `[CODELINE]` represents the major version string (for example, 0.1.0) and `[TEST]` represents the given dataset (ASTEX, RNA or DUD).

2.4.1 Example unpacking procedure

Create a new directory for building RxDock.

```
$ mkdir ~/dev
```

The directory you created is referred to as `[BUILDDIR]` in the subsequent steps.

Copy or download the distribution files to `[BUILDDIR]`.

```
$ cp ~/mydownloads/rxdock-0.1.0.tar.gz ~/dev/
```

Extract the distributions.

```
$ cd ~/dev/
$ tar -xvzf rxdock-0.1.0.tar.gz
```

The distributions contain files with relative path names, and you should find the following subdirectories created under `rxdock-[CODELINE]`. Note that the `./rxdock-0.1.0` subdirectory may have a different name depending on the major version string (see above).

```
$ find . -type d
.
./fw
./src
./src/daylight
./src/lib
./src/exe
./src/GP
./build
./build/test
./build/test/RBT_HOME
./build/tmakelib
./build/tmakelib/linux-pathCC-64
./build/tmakelib/linux-g++-64
./build/tmakelib/linux-g++
./build/tmakelib/unix
./data
./data/filters
./data/sf
```

(continues on next page)

(continued from previous page)

```
./data/pmf
./data/pmf/smoothed
./data/scripts
./lib
./import
./import/tnt
./import/tnt/include
./import/simplex
./import/simplex/src
./import/simplex/include
./docs
./docs/images
./docs/newDocs
./include
./include/GP
./bin
```

Make a note of the following locations for later use.

The RxDock root directory is `[BUILDDIR]/rxdock-[CODELINE]` and will be referred to as `[RBT_ROOT]` in later instructions. In this example, `[RBT_ROOT]` is `~/dev/rxdock-0.1.0/`.

2.5 Building

RxDock is written in C++ (with a small amount of C code from Numerical Recipes) and makes heavy use of the C++ Standard Template Library (STL). The majority of the source code is compiled into a single shared library (`libRbt.so`). The executable programs themselves are relatively light-weight command-line applications linked with `libRbt.so`.

The tmake build systems (from Trolltech) is used to generate makefiles automatically for a particular build target (i.e. combination of operating system and compiler). The source distribution comes with tmake templates defining the compiler options and flags for three Linux build targets (`linux-g++` and `linux-g++-64`). The build targets have been tested under openSUSE 11.3 (2.6.34.10-0.2 kernel) with GNU g++ (versions 3.3.4, 4.5.0, and 4.7.2).

Table 2.3: Standard tmake build targets provided

Target Name	Architecture	Compiler	Compiler flags (release build)
`linux-g++`	32-bit	`g++`	`-m32 -O3 -ffast-math`
`linux-g++-64`	64-bit	`g++`	`-m64 -O3 -ffast-math`

2.5.1 Customising the tmake template for a build target

If none of the tmake templates are suitable for your machine, or if you wish to customise the compiler options, you should first customise one of the existing templates. The tmake template files are stored under `[RBT_ROOT]/build/tmakelib/`. Locate and edit the `tmake.conf` file for the build target you wish to customise. For example, to customise the `linux-g++` build target, edit `[RBT_ROOT]/build/tmakelib/linux-g++/tmake.conf` and localise the values to suit your compiler.

2.5.2 Build procedure

To build RxDock, first go to the `[RBT_ROOT]/build/` directory.

```
$ cd [RBT_ROOT]/build
```

Compile

Make one of the build targets listed below.

```
$ make linux-g++
$ make linux-g++-64
```

Test

Run the RxDock unit tests to check build integrity. If no failed tests are reported you should be all set.

```
$ make test
```

Cleanup (optional)

To remove all intermediate build files from `[RBT_ROOT]/build/`, leaving just the final executables (in `[RBT_ROOT]/bin/`) and shared libraries (in `[RBT_ROOT]/lib/`):

```
$ make clean
```

To remove the final executables and shared libraries as well, returning to a source-only distribution:

```
$ make distclean
```

2.6 Validation experiments

In this section you will find the instructions about how to reproduce our validation experiments using different test sets. Three different sets were analyzed for three different purposes:

- ASTEX set for binding mode prediction in proteins.
- RNA set for assess RNA-ligand docking.
- DUD set for database enrichment.

2.6.1 Binding mode prediction in proteins

First of all, please go to ASTEX set SourceForge download page to download a compressed file with the necessary data.

After downloading the file `ASTEX_rDock_TestSet.tar.gz`, uncompress the file with the following command, which will create a folder called `ASTEX_rDock_TestSet`:

```
$ tar -xvzf ASTEX_rDock_TestSet.tar.gz
$ cd ASTEX_rDock_TestSet/
```

Here you will have the instructions for one of the systems (`1sj0`), to run with the rest of the systems, just change the pdb code with the one desired. Then, make sure that the necessary environmental variables for running RxDock are well defined and run the following commands for entering to the folder and running RxDock with the same settings that we have used:

```
$ cd 1sj0/

#first create the cavity using rbcavity
$ rbcavity -r 1sj0_rdock.prm -was > 1sj0_cavity.log

#then use rbdock to run docking
$ rbdock -r 1sj0_rdock.prm -p dock.prm -n 100 -i 1sj0_ligand.sd \
-o 1sj0_docking_out > 1sj0_docking_out.log

#sdsort for sorting the results according to their score
$ sdsort -n -f'SCORE' 1sj0_docking_out.sd > 1sj0_docking_out_sorted.sd

#calculate rmsd from the output comparing with the crystal structure of the ligand
$ sdrmsd 1sj0_ligand.sd 1sj0_docking_out_sorted.sd
```

2.6.2 Binding mode prediction in RNA

In a similar way of the section above, here you will find a brief tutorial on how to run RxDock with the RNA TestSet used in the validation. As in the first section, please go to RNA set SourceForge download page to download a compressed file with the necessary data.

After downloading the file `RNA_rDock_TestSet.tar.gz`, uncompress the file with the following command, which will create a folder called `RNA_rDock_TestSet`:

```
$ tar -xvzf RNA_rDock_TestSet.tar.gz
$ cd RNA_rDock_TestSet/
```

Here you will have the instructions for one of the systems (`1nem`), to run with the rest of the systems, just change the pdb code with the one desired. Then, make sure that the necessary environmental variables for running RxDock are well defined and run the following commands for entering to the folder and running RxDock with the same settings that we have used (if you have run the previous set, the variables should already be correctly defined):

```
$ cd 1nem/

#first create the cavity using rbcavity
$ rbcavity -r 1nem_rdock.prm -was > 1nem_cavity.log

#then use rbdock to run docking
$ rbdock -r 1nem_rdock.prm -p dock.prm -n 100 -i 1nem_lig.sd \
-o 1nem_docking_out > 1nem_docking_out.log

#sdsort for sorting the results according to their score
$ sdsort -n -f'SCORE' 1nem_docking_out.sd > 1nem_docking_out_sorted.sd

#calculate rmsd from the output comparing with the crystal structure of the ligand
$ sdrmsd 1nem_lig.sd 1nem_docking_out_sorted.sd
```

2.6.3 Database enrichment (actives vs. decoys – for HTVS)

In this section you will find a brief tutorial on how to run RxDock with the DUD test set used in the validation and how to perform different analysis of the results. As in the sections above, please go to DUD set SourceForge download page to download a compressed file with the necessary data.

After downloading the file `DUD_rDock_TestSet.tar.gz`, uncompress the file with the following command, which will create a folder called `DUD_rDock_TestSet`:

```
$ tar -xvzf DUD_rDock_TestSet.tar.gz
$ cd DUD_rDock_TestSet/
```

Here you will have the instructions for one of the systems (`hivpr`), to run with the rest of the systems, just change the DUD system code with the one desired. Then, make sure that the necessary environmental variables for running RxDock are well defined and run the following commands for entering to the folder and running RxDock with the same settings that we have used (if you have run the previous sets, the variables should already be correctly defined):

```
$ cd hivpr/

#first create the cavity using rbcavity
$ rbcavity -r hivpr_rdock.prm -was > hivpr_cavity.log
```

As the number of ligands to dock is very high, we suggest you to use any distributed computing environments, such as SGE or Condor, and configure RxDock to run in multiple CPUs. Namely, split the input ligands file in as many parts as desired (very easy using `sdsplit` tool) and run independent RxDock docking jobs for each "splitted" input file. However, for this example purpose, you will have the instructions for running all set of actives and decoys in one docking job:

```
#uncompress ligand file
$ gunzip hivpr_ligprep.sdf.gz

#use rbdock to run docking
$ rbdock -r hivpr_rdock.prm -p dock.prm -n 100 -i hivpr_ligprep.sdf \
-o hivpr_docking_out > hivpr_docking_out.log

#sdsort with -n and -s flags will sort internally each ligand by increasing
#score and sdfilter will get only the first entry of each ligand
$ sdsort -n -s -fSCORE hivpr_docking_out.sd | sdfilter \
-f'$_COUNT == 1' > hivpr_1poseperlig.sd

#sdreport will print all the scores of the output in a tabular format and,
#with command awk, we will format the results
$ sdreport -t hivpr_1poseperlig.sd | awk '{print $2,$3,$4,$5,$6,$7}' > dataforR_uq.txt
```

At this point, you should have a file called `hivpr_docking_out.sd` with all docking poses written by RxDock (100 * number of input ligands), a file called `hivpr_1poseperlig.sd` with the best scored docking pose for each ligand and a file called `dataforR_uq.txt` that will be used for calculating ROC curves using R. The next step is to calculate ROC curves and other statistics. To do so, please visit section *Calculating ROC curves* and jump to the subsection "R commands for generating ROC curves".

CHAPTER

THREE

REFERENCE GUIDE

In this section you can find the documentation containing full explanation of all RxDock software package and features.

For installation details and first-users instructions, please visit *Installation* and *Getting started* sections.

3.1 Preface

It is intended to develop this document into a full reference guide for the RxDock platform, describing the software tools, parameter files, scoring functions, and search engines. The reader is encouraged to cross-reference the descriptions with the corresponding source code files to discover the finer implementation details.

3.2 Acknowledgements

No third-party C++ libraries or executables are included within the source code. We acknowledge Matthieu Brucher, the author of Audio ToolKit, for his guidance on the implementation of the Nelder-Mead method in C++ using the Eigen library.

3.3 Introduction

The RxDock platform is a suite of command-line tools for high-throughput docking and virtual screening. The programs and methods were developed and validated from 1998 to 2002 at RiboTargets (more recently, Vernalis) for proprietary use. The original program (RiboDock) was designed for high-throughput virtual screening of large ligand libraries against RNA targets, in particular the bacterial ribosome. Since 2002 the programs have been substantially rewritten and validated for docking of drug-like ligands to protein and RNA targets. A variety of experimental restraints can be incorporated into the docking calculation, in support of an integrated structure-based drug design process. In 2006, the software was licensed to the University of York for maintenance and distribution under the name rDock and, in 2012, Vernalis and the University of York agreed to release the program as open-source software.

rDock is licensed under GNU LGPL version 3.0 with support from the University of Barcelona – rdock.sourceforge.net.

Last change to rDock was made in 2014. RxDock is an actively developed fork of rDock started by RxTx in 2019.

3.4 Configuration

Before launching RxDock, make sure the following environment variables are defined. Precise details are likely to be site-specific.

- `RBT_ROOT` environment variable: should be defined to point to the RxDock installation directory.
- `RBT_HOME` environment variable: is optional, but can be defined to to point to a user project directory containing RxDock input files and customised data files.
- `PATH` environment variable: `$RBT_ROOT/bin` should be added to the `$PATH` environment variable.
- `LD_LIBRARY_PATH`: `$RBT_ROOT/lib` should be added to the `$LD_LIBRARY_PATH` environment variable.

3.4.1 Input file locations

The search path for the majority of input files for RxDock is:

- Current working directory
- `$RBT_HOME`, if defined
- The appropriate subdirectory of `$RBT_ROOT/data/`. For example, the default location for scoring function files is `$RBT_ROOT/data/sf/`.

The exception is that input ligand SD files are always specified as an absolute path. If you wish to customise a scoring function or docking protocol, it is sufficient to copy the relevant file to the current working directory or to `$RBT_HOME`, and to modify the copied file.

3.4.2 Launching executables

For small scale experimentation, the RxDock executables can be launched directly from the command line. However, serious virtual screening campaigns will likely need access to a compute farm. In common with other docking tools, RxDock uses the embarrassingly parallel approach to distributed computing. Large ligand libraries are split into smaller chunks, each of which is docked independently on a single machine. Docking jobs are controlled by a distributed resource manager (DRM) such as Condor or SGE.

3.5 Cavity mapping

Virtual screening is very rarely conducted against entire macromolecules. The usual practice is to dock small molecules in a much more confined region of interest. RxDock makes a clear distinction between the region the ligand is allowed to explore (known here as the docking site), and the receptor atoms that need to be included in order to calculate the score correctly. The former is controlled by the cavity mapping algorithm, whilst the latter is scoring function dependent as it depends on the distance range of each component term (for example, vdW range >> polar range). For this reason, it is usual practice with RxDock to prepare intact receptor files (rather than truncated spheres around the region of interest), and to allow each scoring function term to isolate the relevant receptor atoms within range of the docking site.

RxDock provides two methods for defining the docking site:

- Two sphere method
- Reference ligand method

Note: All the keywords found in capital letters in following cavity mapping methods explanation (e.g. `RADIUS`), make reference to the parameters defined in `prm` RxDock configuration file. For more information, go to *Cavity mapping subsection* in *System definition file section*.

3.5.1 Two sphere method

The two sphere method aims to find cavities that are accessible to a small sphere (of typical atomic or solvent radius) but are inaccessible to a larger sphere. The larger sphere probe will eliminate flat and convex regions of the receptor surface, and also shallow cavities. The regions that remain and are accessible to the small sphere are likely to be the nice well defined cavities of interest for drug design purposes.

1. A grid is placed over the cavity mapping region, encompassing a sphere of radius = `RADIUS`, center = `CENTER`. Cavity mapping is restricted to this sphere. All cavities located will be wholly within this sphere. Any cavity that would otherwise protrude beyond the cavity mapping sphere will be truncated at the periphery of the sphere.

   missing figure for step 1

2. Grid points within the volume occupied by the receptor are excluded (coloured red). The radii of the receptor atoms are increased temporarily by `VOL_INCR` in this step.

   missing figure for step 2

3. Probes of radii `LARGE_SPHERE` are placed on each remaining unallocated grid point and checked for clashes with receptor excluded volume. To eliminate edge effects, the grid is extended beyond the cavity mapping region by the diameter of the large sphere (for this step only). This allows the large probe to be placed on grid points outside of the cavity mapping region, yet partially protrude into the cavity mapping region.

   missing figure for step 3

4. All grid points within the cavity mapping region that are accessible to the large probe are excluded (coloured green).

   missing figure for step 4

5. Probes of radii `SMALL_SPHERE` are placed on each remaining grid point and checked for clashes with receptor excluded volume (red) or large probe excluded volume (green).

   missing figure for step 5

6. All grid points that are accessible to the small probe are selected (yellow).

   missing figure for step 6

7. The final selection of cavity grid points is divided into distinct cavities (contiguous regions). In this example only one distinct cavity is found. User-defined filters of `MIN_VOLUME` and `MAX_CAVITIES` are applied at this stage to select a subset of cavities if required. Note that the filters will accept or reject intact cavities only.

   missing figure for step 7

3.5.2 Reference ligand method

The reference ligand method provides a much easier option to define a docking volume of a given size around the binding mode of a known ligand, and is particularly appropriate for large scale automated validation experiments.

1. Reference coordinates are read from `REF_MOL`. A grid is placed over the cavity mapping region, encompassing overlapping spheres of radius = `RADIUS`, centered on each atom in `REF_MOL`. Grid points outside of the overlapping spheres are excluded (coloured green). Grid points within the volume occupied by the receptor are excluded (coloured red). The vdW radii of the receptor atoms are increased by `VOL_INCR` in this step.

   missing figure for step 1

2. Probes of radii `SMALL_SPHERE` are placed on each remaining grid point and checked for clashes with red or green regions.

   missing figure for step 2

3. All grid points that are accessible to the small probe are selected (yellow).

   missing figure for step 3

4. The final selection of cavity grid points is divided into distinct cavities (contiguous regions). In this example only one distinct cavity is found. User-defined filters of `MIN_VOLUME` and `MAX_CAVITIES` are applied at this stage to select a subset of cavities if required. Note that the filters will accept or reject intact cavities only.

   missing figure for step 4

3.6 Scoring functions

3.6.1 Component scoring functions

The RxDock master scoring function (S_{total}) is a weighted sum of intermolecular (S_{inter}), ligand intramolecular (S_{intra}), site intramolecular (S_{site}), and external restraint terms ($S_{\text{restraint}}$) (3.1). S_{inter} is the main term of interest as it represents the protein-ligand (or RNA-ligand) interaction score (3.2). S_{intra} represents the relative energy of the ligand conformation (3.3). Similarly, S_{site} represents the relative energy of the flexible regions of the active site (3.4). In the current implementation, the only flexible bonds in the active site are terminal OH and NH3+ bonds. $S_{\text{restraint}}$ is a collection of non-physical restraint functions that can be used to bias the docking calculation in several useful ways (3.5).

$$S^{\text{total}} = S^{\text{inter}} + S^{\text{intra}} + S^{\text{site}} + S^{\text{restraint}} \tag{3.1}$$

$$\begin{aligned} S^{\text{inter}} = W^{\text{inter}}_{\text{vdW}} \cdot S^{\text{inter}}_{\text{vdW}} + W^{\text{inter}}_{\text{polar}} \cdot S^{\text{inter}}_{\text{polar}} + W^{\text{inter}}_{\text{repul}} \cdot S^{\text{inter}}_{\text{repul}} + W^{\text{inter}}_{\text{arom}} \cdot S^{\text{inter}}_{\text{arom}} + \\ + W_{\text{solv}} \cdot S_{\text{solv}} + W_{\text{rot}} \cdot N_{\text{rot}} + W_{\text{const}} \end{aligned} \tag{3.2}$$

$$S^{\text{intra}} = W^{\text{intra}}_{\text{vdW}} \cdot S^{\text{intra}}_{\text{vdW}} + W^{\text{intra}}_{\text{polar}} \cdot S^{\text{intra}}_{\text{polar}} + W^{\text{intra}}_{\text{repul}} \cdot S^{\text{intra}}_{\text{repul}} + W^{\text{intra}}_{\text{dihedral}} \cdot S^{\text{intra}}_{\text{dihedral}} \tag{3.3}$$

$$S^{\text{site}} = W^{\text{site}}_{\text{vdW}} \cdot S^{\text{site}}_{\text{vdW}} + W^{\text{site}}_{\text{polar}} \cdot S^{\text{site}}_{\text{polar}} + W^{\text{site}}_{\text{repul}} \cdot S^{\text{site}}_{\text{repul}} + W^{\text{site}}_{\text{dihedral}} \cdot S^{\text{site}}_{\text{dihedral}} \tag{3.4}$$

$$S^{\text{restraint}} = W_{\text{cavity}} \cdot S_{\text{cavity}} + W_{\text{tether}} \cdot S_{\text{tether}} + W_{\text{nmr}} \cdot S_{\text{nmr}} + W_{\text{ph4}} \cdot S_{\text{ph4}} \tag{3.5}$$

S_{inter}, S_{intra}, and S_{site} are built from a common set of constituent potentials, which are described below. The main changes to the original RiboDock scoring function [RiboDock2004] are:

i. the replacement of the crude steric potentials (S_{lipo} and S_{rep}) with a true van der Waals potential, S_{vdW}

ii. the introduction of two generalised terms for all short range attractive (S_{polar}) and repulsive (S_{repul}) polar interactions

iii. the implementation of a fast weighed solvent accessible surface (WSAS) area solvation term

iv. the addition of explicit dihedral potentials

van der Waals potential

We have replaced the S_{lipo} and S_{rep} empirical potentials used in RiboDock with a true vdW potential similar to that used by GOLD [GOLD2005]. Atom types and vdW radii were taken from the Tripos 5.2 force field and are listed in the *Appendix section* (Table 3.18). Energy well depths are switchable between the original Tripos 5.2 values and those used by GOLD, which are calculated from the atomic polarisability and ionisation potentials of the atom types involved. Additional atom types were created for carbons with implicit hydrogens, as the Tripos force field uses an all-atom representation. vdW radii for implicit hydrogen types are increased by 0.1 Å for each implicit hydrogen, with well depths unchanged. The functional form is switchable between a softer 4-8 and a harder 6-12 potential. A quadratic potential is used at close range to prevent excessive energy penalties for atomic clashes. The potential is truncated at longer range ($1.5 \cdot r_{\min}$, the sum of the vdW radii).

The quadratic potential is used at repulsive energies between e_{cutoff} and e_0, where e_{cutoff} is defined as a multiple of the energy well depth ($e_{\text{cutoff}} = \text{ECUT} \cdot e_{\min}$), and e_0 is the energy at zero separation, defined as a multiple of e_{cutoff} ($e_0 = \text{E0} \cdot e_{\text{cutoff}}$). ECUT can vary between 1 and 120 during the docking search (see *Genetic algorithm subsection* in *Docking protocol section*), whereas E0 is fixed at 1.5.

Empirical attractive and repulsive polar potentials

We continue to use an empirical Bohm-like potential to score hydrogen-bonding and other short-range polar interactions. The original RiboDock polar terms ($S_{\text{H-bond}}$, $S_{\text{posC-acc}}$, $S_{\text{acc-acc}}$, $S_{\text{don-don}}$) are generalised and condensed into two scoring functions, S_{polar} and S_{repul} ((3.6) and (3.7), also taking into account (3.8), (3.9), (3.10), (3.11), (3.12), (3.13), (3.14), (3.15), (3.16), (3.17), (3.18), (3.19), and (3.20)), which deal with attractive and repulsive interactions respectively. Six types of polar interaction centres are considered: hydrogen bond donors (DON), metal ions (M+), positively charged carbons (C+, as found at the centre of guanidinium, amidinium and imidazole groups), hydrogen bond acceptors with pronounced lone pair directionality (ACC_LP), acceptors with in-plane preference but limited lone-pair directionality (ACC_PLANE), and all remaining acceptors (ACC). The ACC_LP type is used for carboxylate oxygens and O sp2 atoms in RNA bases, with ACC_PLANE used for other O sp2 acceptors. This distinction between acceptor types was not made in RiboDock, in which all acceptors were implicitly of type ACC.

$$S_{\text{polar}} = \sum_{\text{IC1-IC2}} f_1(|\Delta R_{12}|) \cdot \text{ANG}_{\text{IC1}} \cdot \text{ANG}_{\text{IC2}} \cdot f_2(\text{IC1}) \cdot f_2(\text{IC2}) \cdot f_3(\text{IC1}) \cdot f_3(\text{IC2}) \tag{3.6}$$

$$S_{\text{repul}} = \sum_{\text{IC1-IC2}} f_1(\Delta R_{12}) \cdot \text{ANG}_{\text{IC1}} \cdot \text{ANG}_{\text{IC2}} \cdot f_2(\text{IC1}) \cdot f_2(\text{IC2}) \cdot f_3(\text{IC1}) \cdot f_3(\text{IC2}) \tag{3.7}$$

$$f_1(\Delta X) = \begin{cases} 1 & \Delta X \leq \Delta X_{\min} \\ 1 - \frac{\Delta X - \Delta X_{\min}}{\Delta X_{\max} - \Delta X_{\min}} & \Delta X_{\min} < \Delta X \leq \Delta X_{\max} \\ 0 & \Delta X > \Delta X_{\max} \end{cases} \tag{3.8}$$

$$f_2(i) = sgn(i)(1 + 0.5|c_i|) \tag{3.9}$$

$$sgn(i) = \begin{cases} -1 & \text{ACC, ACC\_LP, ACC\_PLANE} \\ +0.5 & \text{C+} \\ +1.0 & \text{DON, M+} \end{cases} \tag{3.10}$$

$$c_i = \text{formal charge on primary atom of interaction centre } i \tag{3.11}$$

$$f_3(\Delta X) = \begin{cases} \sqrt{\frac{N_i}{25}} & \text{macromolecular interaction centres} \\ 1 & \text{ligand interaction centres} \end{cases} \tag{3.12}$$

$$N_i = \text{number of non-hydrogen macromolecule atoms within 5 Å radius of primary atom of interaction centre } i \tag{3.13}$$

Individual interaction scores are the product of simple scaling functions for geometric variables, formal charges and local neighbour density. The scaling functions themselves, and the formal charge assignment method, are retained from

RiboDock. Metals are assigned a uniform formal charge of +1. C+ is considered to be a weak donor in this context and scores are scaled by 50 % relative to conventional donors by the assignment of $sgn(i) = 0.5$ in (3.9). Pseudo-formal charges are no longer assigned to selected RNA base atoms. The geometric functions minimally include an interaction distance term, with the majority also including angular terms dependent on the type of the interaction centres. Geometric parameters and the angular functions are summarised in *Appendix section* (Table 3.19 and Table 3.20, respectively).

The most notable improvements to RiboDock are that attractive (hydrogen bond and metal) interactions with ACC_LP and ACC_PLANE acceptors include terms for ϕ and θ (as defined in [ref 3]) to enforce the relevant lone pair directionality. These replace the α_{ACC} dependence, which is retained for the ACC acceptor type. No distinction between acceptor types is made for attractive interactions with C+ carbons, or for repulsive interactions between acceptors. In these circumstances all acceptors are treated as type ACC. Such C+-ACC interactions, which in RiboDock were described by only a distance function, ($S_{\text{posC-acc}}$) now include angular functions around the carbon and acceptor groups. Repulsive interactions between donors, and between acceptors, also have an angular dependence. This allows a stronger weight, and a longer distance range, to be used to penalise disallowed head-to-head interactions without forbidding allowable contacts. One of the issues in RiboDock was that it was not possible to include neutral acceptors in the acceptor-acceptor repulsion term with a simple distance function.

Solvation potential

The desolvation potential in RxDock combines a weighted solvent-accessible surface area approach [WSAS2001] with a rapid probabilistic approximation to the calculation of solvent-accessible surface areas [RASASA1988] based on pairwise interatomic distances and radii ((3.14), taking into account (3.15), (3.16), (3.17), (3.18), (3.19), and (3.20)).

$$S_{\text{solv}} = (\Delta G_{\text{WSAS}}^{\text{site,bound}} - \Delta G_{\text{WSAS}}^{\text{site}_0,\text{unbound}}) + (\Delta G_{\text{WSAS}}^{\text{ligand,bound}} - \Delta G_{\text{WSAS}}^{\text{ligand}_0,\text{unbound}}) \tag{3.14}$$

$$r_{\text{s}} = 0.6\text{Å} \tag{3.15}$$

$$p_{ij} = \begin{cases} 0.8875 & \text{1-2 intramolecular connections} \\ 0.3516 & \text{1-3 intramolecular connections} \\ 0.3156 & \text{1-4 intramolecular connections and above} \\ 0.3156 & \text{intermolecular interactions} \end{cases} \tag{3.16}$$

$$S_i = 4\pi(r_i + r_s)^2 \tag{3.17}$$

$$b_{ij} = \pi(r_i + r_s)(r_j + r_i + 2r_s - d)\Big(1 - \frac{r_j - r_i}{d}\Big) \tag{3.18}$$

$$A_i = S_i \prod_j 1 - \frac{p_i p_{ij} b_{ij}}{S_i} \tag{3.19}$$

$$\Delta G_{\text{WSAS}} = \sum_{i=1}^{n_i} w_i A_i \tag{3.20}$$

The calculation is fast enough therefore to be used in docking. We have redefined the solvation atom types compared to the original work [WSAS2001] and recalibrated the weights against the same training set of experimental solvation free energies in water (395 molecules). The total number of atom types (50, including 6 specifically for ionic groups and metals) is slightly lower than in original work (54). Our atom types reflect the fact that RxDock uses implicit non-polar hydrogens. The majority of types are a combination of hybridisation state and the number of implicit or explicit hydrogens. All solvation parameters are listed in *Appendix section* (Table 3.21).

S_{solv} is calculated as the change in solvation energy of the ligand and the docking site upon binding of the ligand. The reference energies are taken from the initial conformations of the ligand and site (as read from file) and not from the current pose under evaluation. This is done to take into account any changes to intramolecular solvation energy. Strictly speaking the intramolecular components should be reported separately under S_{intra} and S_{site} but this is not done for reasons of computational efficiency.

Dihedral potential

Dihedral energies are calculated using Tripos 5.2 dihedral parameters for all ligand and site rotatable bonds. Corrections are made to account for the missing contributions from the implicit non-polar hydrogens.

3.6.2 Intermolecular scoring functions under evaluation

Training sets

We constructed a combined set of protein-ligand and RNA-ligand complexes for training of RxDock. Molecular data files for the protein-ligand complexes were extracted from the downloaded CCDC/Astex cleanlist [ASTEX2007] and used without substantive modification. The only change was to convert ligand MOL2 files to MDL SD format using Corina [CORINA1990], leaving the coordinates and protonation states intact.

Protein MOL2 files were read directly. The ten RNA-ligand NMR structures from the RiboDock validation set were supplemented with five RNA-ligand crystal structures (1f1t, 1f27, 1j7t, 1lc4, 1mwl) prepared in a similar way. All 15 RNA-ligand structures have measured binding affinities.

58 complexes (43 protein-ligand and 15 RNA-ligand) were selected for the initial fitting of component scoring function weights. Protein-ligand structures were chosen (of any X-ray resolution) that had readily available experimental binding affinities [PDBbind2004]. 102 complexes were used for the main validation of native docking accuracy for different scoring function designs, consisting of 87 of the 92 entries in the high resolution (R < 2 Å) clean-list (covalently bound ligands removed – 1aec, 1b59, 1tpp, 1vgc, 4est), and the 15 RNA-ligand complexes.

Scoring functions design

Component weights (W) for each term in the intermolecular scoring function (S_{inter}) were obtained by least squares regression of the component scores to ΔG_{bind} values for the binding affinity training set described above (58 entries). Each ligand was subjected first to simplex minimisation in the docking site, starting from the crystallographic pose, to relieve any minor non-bonded clashes with the site. The scoring function used for minimisation was initialised with reasonable manually assigned weights. If the fitted weights deviated significantly from the initial weights the procedure was repeated until convergence. Certain weights (W_{repul}, W_{rot}, W_{const}) were constrained to have positive values to avoid non-physical, artefactual models. Note that the presence of W_{rot} and W_{const} in S_{inter} improves the quality of the fit to the binding affinities but does not impact on native ligand docking accuracy.

Ten intermolecular scoring functions were derived with various combinations of terms (Table 3.1). SF0 is a baseline scoring function that has the van der Waals potential only. SF1 adds a simplified polar potential, without f2 (formal charge) and f3 (neighbour density) scaling functions, and with a single acceptor type (ACC) that lacks lone-pair directionality. SF2 has the full polar potential (f2 and f3 scaling functions, ACC, ACC_LP and ACC_PLANE acceptor types) and adds the repulsive polar potential. SF3 has the same functional form as SF2 but with empirical weights in regular use at RiboTargets. SF4 replaces the repulsive polar potential with the WSAS desolvation potential described above. SF5 has the same functional form as SF4 but with empirical weights in regular use at RiboTargets. SF6 combines the repulsive polar and desolvation potentials. SF7 has the same functional form as SF2 and SF3 but with weights for W_{vdW} and W_{polar} taken from SF5. SF8 and SF9 add the crude aromatic term from RiboDock [RiboDock2004] to SF3 and SF5 respectively. The S_{intra} functional form and weights were held constant, and equivalent to SF3, to avoid any differences in ligand conformational energies affecting the docking results. As the S_{site} scores are calculated simultaneously with S_{inter} (for computational reasons) the S_{site} functional form and weights vary in line with S_{inter}. There is surprisingly little variation in correlation coefficient (R) and root mean square error (RMSE) in predicted binding energy over the ten scoring functions (Table 3.1). The best results are obtained with SF4 (R = 0.67, RMSE = 9.6 kJ/mol).

Table 3.1: Intermolecular scoring function weights under evaluation (a = constrained to be > zero; b = fixed values; c = correlation coefficient (R), and root mean squared error (RMSE) between S_{inter} and ΔG_{bind}, for minimised experimental ligand poses, over binding affinity validation set (58 entries)).

SF	W_{vdW}	W_{polar}	W_{solv}	W_{repul}[a]	W_{arom}	W_{rot}[a]	W_{const}[a]	R[c]	RMSE[c]
0	1.4	-	-	-	-	0	0	0.62	10.9
1	1.126	2.36	-	-	-	0.217	0	0.64	10.2
2	1.192	2.087	-	2.984	-	0	0	0.63	10.4
3	1.000[b]	3.400[b]	-	5.000[b]	-	0	0	0.59	10.9
4	1.317	3.56	0.449	-	-	0	4.	0.67	9.6
5	1.500[b]	5.000[b]	0.500[b]	-	-	0.568	4.782	0.62	10.7
6	1.314	4.447	0.500[b]	5.000[b]	-	0	0	0.62	10.4
7	1.500[b]	5.000[b]	-	5.000[b]	-	0.986	12.046	0.55	12.9
8	1.000[b]	3.400[b]	-	5.000[b]	-1.6[b]	0	0	0.53	11.8
9	1.500[b]	5.000[b]	0.500[b]	-	-1.6[b]	0.647	5.056	0.58	11.5

Scoring functions validation

The ability of the ten intermolecular scoring functions (SF0 to SF9) to reproduce known ligand binding modes was determined on the combined test set of 102 protein-ligand and RNA-ligand complexes. The intra-ligand scoring function (S_{intra}) was kept constant, with component weights equivalent to SF3, and a dihedral weight of 0.5. Terminal OH and NH3 groups on the receptor in the vicinity of the docking site were fully flexible during docking. Ligand pose populations of size $N_{\text{pop}} = 300$ were collected for each complex and intermolecular scoring function combination. The population size was increased to $N_{\text{pop}} = 1000$ for two of the most promising scoring functions (SF3 and SF5).

Protein-ligand docking accuracy is remarkably insensitive to scoring function changes. Almost half of the ligand binding modes can be reproduced with a vdW potential only (SF0). The addition of a simplified polar potential (SF1) increases the accuracy to over 70 % of protein-ligand test cases predicted to within 2 Å RMSD. The success rate increases further to 78 % with SF3, which has the full attractive and repulsive polar potentials, and empirically adjusted weights relative to SF2. Subsequent changes to the component terms and weights, including the addition of the desolvation potential, have little or no impact on the protein-ligand RMSD metric.

The nucleic acid set shows a much greater sensitivity to scoring function changes. This can in part be explained by the smaller sample size that amplifies the percentage changes in the RMSD metric, but nevertheless the trends are clear. There is a gradual increase in docking accuracy from SF0 (37 %) to SF3 (52 %), but absolute performance is much lower than for the protein-ligand test set. This level of docking accuracy for nucleic acid-ligand complexes is broadly consistent with the original RiboDock scoring function, despite the fact that the original steric term (LIPO) has been replaced by a true vdW potential. The introduction of the desolvation potential in place of the empirical repulsive polar potential (in SF4 and SF5) results in a substantial improvement in accuracy, to around 70 % of test cases within 2 Å RMSD. Subsequent changes (SF6 to SF9) degrade the accuracy. The lower performance of SF7, which has the higher weights for the VDW and POLAR terms taken from SF5 but lacks the desolvation potential, demonstrates that it is the desolvation term itself that is having the beneficial effect, and not merely the reweighting of the other terms. The inclusion of the geometric aromatic term in SF8 and SF9 has a detrimental impact on the performance of SF3 and SF5 respectively.

Overall, SF5 achieves optimum performance across proteins and nucleic acids (76.7 % within 2 Å RMSD). SF3 (no desolvation potential) and SF5 (with desolvation potential) were selected as the best intermolecular scoring functions. Finally, these two scoring functions, SF3 and SF5, were the ones implemented in RxDock with the names of `dock.prm` and `dock_solv.prm`, respectively.

Note: In virtual screening campaigns, or in experiments where score of different ligands is compared, the best scoring poses for each molecule (as defined by the lowest S_{total} within the sample) are sorted and ranked by S_{inter}. In other

words, the contributions to S_{total} from S_{intra}, S_{site} and $S_{\text{restraint}}$ are ignored when comparing poses between different ligands against the same target. The rationale for this is that, in particular, the ligand intramolecular scores are not on an absolute scale and can differ markedly between different ligands.

3.6.3 Code implementation

Scoring functions for docking are constructed at run-time (by `RbtSFFactory` class) from scoring function definition files (RxDock `.prm` format). The default location for scoring function definition files is `$RBT_ROOT/data/sf/`.

The total score is an aggregate of intermolecular ligand-receptor and ligand-solvent interactions (branch `SCORE.INTER`), intra-ligand interactions (branch `SCORE.INTRA`), intra-receptor, intra-solvent and receptor-solvent interactions (branch `SCORE.SYSTEM`), and external restraint penalties (branch `SCORE.RESTR`).

The `SCORE.INTER`, `SCORE.INTRA` and `SCORE.SYSTEM` branches consist of weighted sums of interaction terms as shown below. Note that not all terms appear in all branches. See the rDock draft paper [rDock2014] for more details on the implementation of these terms.

Table 3.2: Scoring function terms and C++ implementation classes.

Term	Description	`INTER`	`INTRA`	`SYSTEM`
VDW	van der Waals	`RbtVdWIdxSF`	`RbtVdwIntraSF`	`RbtVdwIdxSF`
VDW	van der Waals (grid based)	`RbtVdwGridSF`	N/A	N/A
POLAR	Attractive polar	`RbtPolarIdxSF`	`RbtPolarIntraSF`	`RbtPolarIdxSF`
REPUL	Repulsive polar	`RbtPolarIdxSF`	`RbtPolarIntraSF`	`RbtPolarIdxSF`
SOLV	Desolvation	`RbtSAIdxSF`	`RbtSAIdxSF`	`RbtSAIdxSF`
CONST	Translation/rotational binding entropy penalty	`RbtConstSF`	N/A	`RbtConstSF`
ROT	Torsional binding entropy penalty	`RbtRotSF`	N/A	N/A

Two intermolecular scoring functions (`SCORE.INTER` branch) have been validated. These are known informally as the standard scoring function and the desolvation scoring function (referred to as SF3 and SF5 respectively in the rDock draft paper [rDock2014]). The standard intermolecular scoring function consists of VDW, POLAR and REPUL terms. In the desolvation scoring function, the REPUL term is replaced by a more finely parameterised desolvation potential (SOLV term) based on a weighted solvent-accessible surface (WSAS) area model. The ligand intramolecular scoring function (`SCORE.INTRA` branch) remains constant in both cases, and has the same terms and weights as the standard intermolecular scoring function.

Table 3.3: Scoring function data files.

File	Description
`RbtInterIdxSF.prm`	Intermolecular scoring function definition (standard scoring function, SF3)
`RbtInterGridSF.prm`	As above, but vdW term uses a precalculated grid
`RbtSolvIdxSF.prm`	Intermolecular scoring function definition (desolvation scoring function, SF5)
`calcgrid_vdw1.prm`	vdW term only (`ECUT = 1`), for calculating vdW grid (used by `rbcalcgrid`)
`calcgrid_vdw5.prm`	vdW term only (`ECUT = 5`), for calculating vdW grid (used by `rbcalcgrid`)
`Tripos52_vdw.prm`	vdW term parameter file
`Tripos52_dihedrals.prm`	Dihedral term parameter file
`solvation_asp.prm`	Desolvation term parameter file

Note: External restraint penalty terms are defined by the user in the system definition `.prm` file. Originally, rDock did not support flexible receptor dihedrals or explicit structural waters, and the overall scoring function consisted of just the `SCORE.INTER` and `SCORE.INTRA` branches. At that time, the intermolecular scoring function definition file (e.g. `RbtInterIdxSF.prm`) represented precisely the `SCORE.INTER` terms, and the intramolecular definition file (`RbtIntraSF.prm`) represented precisely the `SCORE.INTRA` terms. With the introduction of receptor flexibility and explicit structural waters (and hence the need for the `SCORE.SYSTEM` branch), the situation is slightly more complex. For implementation reasons, many of the terms reported under `SCORE.SYSTEM` (with the exception of the dihedral term) are calculated simultaneously with the terms reported under `SCORE.INTER`, and hence their parameterisation is defined implicitly in the intermolecular scoring function definition file. In contrast, the ligand intramolecular scoring function terms can be controlled independently.

3.6.4 References

3.7 Docking protocol

3.7.1 Protocol summary

Pose generation

RxDock uses a combination of stochastic and deterministic search techniques to generate low energy ligand poses. The standard docking protocol to generate a single ligand pose uses 3 stages of genetic algorithm search (GA1, GA2, GA3), followed by low temperature Monte Carlo (MC) and simplex minimisation (MIN) stages.

Several scoring function parameters are varied between the stages to promote efficient sampling. The ECUT parameter of the S_{inter} vdW potential (defining the hardness of the intermolecular close range potential) is increased from 1 in the first GA stage (GA1) to a maximum of 120 in the MC and MIN stages, with intermediate values of 5 in GA2 and 25 in GA3. The functional form of the S_{inter} vdW potential is switched from a 4-8 potential in GA1 and GA2 to a 6-12 potential in GA3, MC and MIN.

In a similar fashion, the overall weight of the S_{intra} dihedral potential is ramped up from an initial value of 0.1 in GA1 to a final value of 0.5 in the MC and MIN stages, with intermediate values of 0.2 in GA2 and 0.3 in GA3. In contrast, the S_{intra} vdW parameters (as used for the ligand intramolecular potential) remain fixed at the final, hard values throughout the calculation (ECUT = 120, 6-12 potential).

Overall, we found this combination of parameter changes allows for efficient sampling of the very poor starting poses, whilst minimising the likelihood that poor ligand internal conformations are artificially favoured and trapped early in the search, and ensures that physically realistic potentials are used for final optimisation and analysis.

Genetic algorithm

The GA chromosome consists of the ligand centre of mass (com), the ligand orientation, as represented by the quaternion (q) required to rotate the ligand principal axes from the Cartesian reference axes, the ligand rotatable dihedral angles, and the receptor rotatable dihedral angles. The ligand centre of mass and orientation descriptors, although represented by multiple floating point values (com.x, com.y, com.z, and q.s, q.x, q.y, q.z respectively), are operated on as intact entities by the GA mutation and crossover operators.

For so-called free docking, in which no external restraints other than the cavity penalty are imposed, the initial population is generated such that the ligand centre of mass is constrained to lie on a randomly selected grid point within the defined docking volume, and the ligand orientation and all dihedral angles are randomised completely. Mutations to the ligand centre of mass are by a random distance along a randomly oriented unit vector. Mutations to the ligand orientation are performed by rotating the ligand principal axes by a random angle around a randomly oriented unit

vector. Mutations to the ligand and receptor dihedral angles are by a random angle. All mutation distances and angles are randomly selected from rectangular distributions of defined width.

A generation is considered to have passed when the number of new individuals created is equal to the population size. Instead of having a fixed number of generations, the GA is allowed to continue until the population converges. The population is considered converged when the score of the best scoring pose fails to improve by more than 0.1 over the last three generations. This allows early termination of poorly performing runs for which the initial population is not able to generate a good solution.

During initial testing the impact of a wide variety of GA parameters (Table 3.4) were explored on a small, representative set of protein-ligand complexes (3ptb, 1rbp, 1stp, 3dfr). We measured the frequency that the algorithm was able to find the experimental conformation, and the average run time. Optimum results were obtained with a steady state GA, roulette wheel selection, a single population of size ($100 \times$ (number of rotatable bonds)), a crossover:mutation ratio of 40:60, and mutation distribution widths of ligand translation 2 Å, ligand rotation of 30 degrees and dihedral angle of 30 degrees. These parameters have been found to be generally robust across a wide variety of systems.

Table 3.4: Summary of GA parameter space explored, and final values

Parameter	Values Explored	Final Values
Number of populations	1, 2, 3, 4, 5	1
Selection operator	Roulette wheel, Rank	Roulette wheel
Mutation	Rectangular Cauchy	Rectangular
GA	Generational, Steady state	Steady state
Elitism	Yes, No	No
No of individuals modified in each generation	All values from 1 to population size	0.5 * population size
Population size	50, 75, 100, 125, 150, 200, 400, 800 * number of rotatable bonds	100 * number of rotatable bonds
Probability of choosing Crossover vs. Mutation	0.0, 0.05, 0.1 … 0.9, 0.95, 1.0	0.4
Torsion step	3, 12, 21, 30 degrees	30 degrees
Rotational step	3, 12, 21, 30 degrees	30 degrees
Translation step	0.1, 0.8, 1.4, 2.0 Å	2.0 Å

Monte Carlo

The method and parameters for low temperature Monte Carlo are similar to those described for phase 4 of the RiboDock simulated annealing search protocol. The overall number of trials is scaled according to the number of rotatable bonds in the ligand, from a minimum of 500 ($N_{\text{rot}} = 0$) to a maximum of 2000 ($N_{\text{rot}} = 15$). Maximum step sizes are: translation 0.1 Å, ligand rotation of 10 degrees and dihedral angle of 10 degrees. Step sizes are halved if the Metropolis acceptance rate falls below 0.25.

Simplex

The Nelder-Mead's simplex minimisation routine operates on the same chromosome representation as the GA, with the exception that the composite descriptors (centre of mass and orientation) are decomposed into their constituent floating point values.

3.7.2 Code implementation

Docking protocols are constructed at run-time (by `RbtTransformFactory` class) from docking protocol definition files (RxDock .prm format). The default location for docking protocol files is `$RBT_ROOT/data/scripts/`. The docking protocol definition file defines the sequence of search algorithms that constitute a single docking run for a single ligand record. Each search algorithm component operates either on a single chromosome representing the system degrees of freedom, or on a population of such chromosomes. The chromosome is constructed (by `RbtChromFactory` class) as an aggregate of individual chromosome elements for the receptor, ligand and explicit solvent degrees of freedom, as defined by the flexibility parameters in the system definition file.

Table 3.5: Chromosome elements

Element	Defined by	Class	Length
Position	Centre of mass	`RbtChromPositionElement`	3
Orientation	Euler angles for principal axes	`RbtChromPositionElement`	3
Dihedral	Dihedral angle for rotatable bond	`RbtChromDihedralElement`	1 per bond
Occupancy	Explicit water occupancy state	`RbtChromOccupancylElement`	1 per water

3.7.3 Standard RxDock docking protocol (`dock.prm`)

As stated above in this section, the standard RxDock docking protocol consists of three phases of a genetic algorithm search, followed by low-temperature Monte Carlo and simplex minimisation.

Table 3.6: Search algorithm components and C++ implementation classes

Component	Class	Operates on
Randomise population	`RbtRandPopTransform`	Chromosome population
Genetic algorithm search	`RbtGATransform`	Chromosome population
Monte Carlo simulated annealing	`RbtSimAnnTransform`	Single chromosome
Simplex minimisation	`RbtSimplexTransform`	Single chromosome
Null operation	`RbtNullTransform`	N/A

Table 3.7: Docking protocol data files

File	Description
`score.prm`	Calculates score only for initial conformation (standard scoring function)
`scole_solv.prm`	As above, but uses desolvation scoring function
`minimise.prm`	Simplex minimisation of initial conformation (standard scoring function)
`minimise_solv.prm`	As above, but uses desolvation scoring function
`dock.prm`	Full docking search (standard scoring function)
`dock_solv.prm`	As above, but uses desolvation scoring function
`dock_grid.prm`	Full docking search (standard scoring function, grid-based vdW term)
`dock_solv_grid.prm`	Full docking search (desolvation scoring function, grid-based vdW term)

By way of example, the `dock.prm` script is documented in detail. The other scripts are very similar.

```
SECTION SCORE
   INTER RbtInterIdxSF.prm
   INTRA RbtIntraSF.prm
   SYSTEM RbtTargetSF.prm
END_SECTION
```

Scoring Function The scoring function definition is referenced within the docking protocol definition file itself, in the SCORE section. This section contains entries for the INTER, INTRA and SYSTEM scoring function definition files.

```
SECTION SETSLOPE_1
   TRANSFORM RbtNullTransform
   # Dock with a high penalty for leaving the cavity
   WEIGHT@SCORE.RESTR.CAVITY 5.0
   # Gradually ramp up dihedral weight from 0.1-->0.5
   WEIGHT@SCORE.INTRA.DIHEDRAL 0.1
   # Gradually ramp up energy cut off for switching to quadratic
   ECUT@SCORE.INTER.VDW 1.0
   # Start docking with a 4-8 vdW potential
   USE 4_8@SCORE.INTER.VDW TRUE
   # Broader angular dependence
   DA1MAX@SCORE.INTER.POLAR 180.0
   # Broader angular dependence
   DA2MAX@SCORE.INTER.POLAR 180.0
   # Broader distance range
   DR12MAX@SCORE.INTER.POLAR 1.5
END_SECTION
```

Genetic algorithm All sections that contain the TRANSFORM parameter are interpreted as a search algorithm component. The value of the TRANSFORM parameter is the C++ implementation class name for that transform. An RbtNullTransform can be used to send messages to the scoring function to modify key scoring function parameters in order to increase search efficiency. All parameter names that contain the @ symbol are interpreted as scoring function messages, where the string before the @ is the scoring function parameter name, the string after the @ is the scoring function term, and the parameter value is the new value for the scoring function parameter. Messages are sent blind, with no success feedback, as the docking protocol has no knowledge of the composition of the scoring function terms.

Here, we start the docking with a soft 4-8 vdW potential, a reduced dihedral potential, and extended polar ranges (distances and angles) for the intermolecular polar potential. These changes are all designed to aid sampling efficiency by not overpenalising bad contacts in the initial, randomised population, and by encouraging the formation of intermolecular hydrogen bonds.

```
SECTION RANDOM_POP
   TRANSFORM RbtRandPopTransform
   POP_SIZE 50
   SCALE_CHROM_LENGTH TRUE
END_SECTION
```

Creates an initial, randomised chromosome population. If SCALE_CHROM_LENGTH is false, the population is of fixed size, defined by POP_SIZE. If SCALE_CHROM_LENGTH is true, the population is proportional to the overall chromosome length, defined by POP_SIZE multiplied by the chromosome length.

```
SECTION GA_SLOPE1
   TRANSFORM RbtGATransform
   PCROSSOVER 0.4 # Prob. of crossover
   XOVERMUT TRUE # Cauchy mutation after each crossover
   CMUTATE FALSE # True = Cauchy; False = Rectang. for regular mutations
   STEP_SIZE 1.0 # Max relative mutation
END_SECTION
```

First round of GA.

```
SECTION SETSLOPE_3
   TRANSFORM RbtNullTransform
```

(continues on next page)

(continued from previous page)

```
   WEIGHT@SCORE.INTRA.DIHEDRAL 0.2
   ECUT@SCORE.INTER.VDW 5.0
   DA1MAX@SCORE.INTER.POLAR 140.0
   DA2MAX@SCORE.INTER.POLAR 140.0
   DR12MAX@SCORE.INTER.POLAR 1.2
END_SECTION
```

Increases the ligand dihedral weight, increases the short-range intermolecular vdW hardness (ECUT), and decreases the range of the intermolecular polar distances and angles.

```
SECTION GA_SLOPE3
   TRANSFORM RbtGATransform
   PCROSSOVER 0.4 # Prob. of crossover
   XOVERMUT TRUE # Cauchy mutation after each crossover
   CMUTATE FALSE # True = Cauchy ; False = Rectang. for regular mutations
   STEP_SIZE 1.0 # Max relative mutation
END_SECTION
```

Second round of GA with revised scoring function parameters.

```
SECTION SETSLOPE_5
   TRANSFORM RbtNullTransform
   WEIGHT@SCORE.INTRA.DIHEDRAL 0.3
   ECUT@SCORE.INTER.VDW 25.0
   # Now switch to a convential 6-12 for final GA, MC, minimisation
   USE 4_8@SCORE.INTER.VDW FALSE
   DA1MAX@SCORE.INTER.POLAR 120.0
   DA2MAX@SCORE.INTER.POLAR 120.0
   DR12MAX@SCORE.INTER.POLAR 0.9
END_SECTION
```

Further increases the ligand dihedral weight, further increases the short-range intermolecular vdW hardness (ECUT), and further decreases the range of the intermolecular polar distances and angles. Also switches from softer 4-8 vdW potential to a harder 6-12 potential for final round of GA, MC and minimisation.

```
SECTION GA_SLOPE5
   TRANSFORM RbtGATransform
   PCROSSOVER 0.4 # Prob. of crossover
   XOVERMUT TRUE # Cauchy mutation after each crossover
   CMUTATE FALSE # True = Cauchy ; False = Rectang. for regular mutations
   STEP_SIZE 1.0 # Max relative mutation
END_SECTION
```

Final round of GA with revised scoring function parameters.

```
SECTION SETSLOPE_10
   TRANSFORM RbtNullTransform
   WEIGHT@SCORE.INTRA.DIHEDRAL 0.5 # Final dihedral weight matches SF file
   ECUT@SCORE.INTER.VDW 120.0 # Final ECUT matches SF file
   DA1MAX@SCORE.INTER.POLAR 80.0
   DA2MAX@SCORE.INTER.POLAR 100.0
   DR12MAX@SCORE.INTER.POLAR 0.6
END_SECTION
```

Resets all the modified scoring function parameters to their final values, corresponding to the values in the scoring function definition files. It is important that the final scoring function optimised by the docking search can be compared directly with the score-only and minimisation-only protocols, in which the scoring function parameters are not

modified.

```
SECTION MC_10K
   TRANSFORM RbtSimAnnTransform
   START_T 10.0
   FINAL_T 10.0
   NUM_BLOCKS 5
   STEP_SIZE 0.1
   MIN_ACC_RATE 0.25
   PARTITION_DIST 8.0
   PARTITION_FREQ 50
   HISTORY_FREQ 0
END_SECTION
```

Monte Carlo Low temperature Monte Carlo sampling, starting from fittest chromosome from final round of GA.

```
SECTION SIMPLEX
   TRANSFORM RbtSimplexTransform
   MAX_CALLS 200
   NCYCLES 20
   STOPPING_STEP_LENGTH 10e-4
   PARTITION_DIST 8.0
   STEP_SIZE 1.0
   CONVERGENCE 0.001
END_SECTION
```

Minimisation Simplex minimisation, starting from fittest chromosome from low temperature Monte Carlo sampling.

```
SECTION FINAL
   TRANSFORM RbtNullTransform
   WEIGHT@SCORE.RESTR.CAVITY 1.0 # revert to standard cavity penalty
END_SECTION
```

Finally, we reset the cavity restraint penalty to 1. The penalty has been held at a value of 5 throughout the search, to strongly discourage the ligand from leaving the docking site.

3.8 System definition file

Although known previously as the receptor `.prm` file, the system definition file has evolved to contain much more than the receptor information. The system definition file is used to define:

- Receptor input files and flexibility parameters (the section called *Receptor definition*)
- Explicit solvent input file and flexibility parameters (the section called *Solvent definition*)
- Ligand flexibility parameters (the section called *Ligand definition*).
- External restraint terms to be added to the total scoring function (e.g. *cavity restraint*, *pharmacophoric restraint*)

3.8.1 Receptor definition

The receptor can be loaded from a single MOL2 file, or from a combination of Charmm PSF and CRD files. In the former case the MOL2 file provides the topology and reference coordinates simultaneously, whereas in the latter case the topology is loaded from the PSF file and the reference coordinates from the CRD file. For historical compatibility reasons, receptor definition parameters are all defined in the top-level namespace and should not be placed between `SECTION` / `END_SECTION` pairs.

Caution: If MOL2 and PSF/CRD parameters are defined together, the MOL2 parameters take precedence and are used to load the receptor model.

Table 3.8: Receptor definition parameters

Parameter	Description	Type	Default	Range of values
Parameters specific to loading receptor in MOL2 file format				
`RECEPTOR_FILE`	Name of receptor MOL2 file	Filename string	No default value	Valid MOL2 filename
Parameters specific to loading receptor in Charmm PSF/CRD file format				
`RECEPTOR_PSF_FILE`	Name of receptor Charmm PSF file	Filename string	No default value	Valid Charmm PSF filename
`RECEPTOR_CRD_FILE`	Name of receptor Charmm CRD file	Filename string	No default value	Valid Charmm CRD filename
`RECEPTOR_MASS_FILE`	Name of RxDock-annotated Charmm masses file	Filename string	No default value	`masses.rtf`, `top_all2_prot_na.inp`
General receptor parameters, applicable to either file format				
`RECEPTOR_SEGMENT_LIST`	List of molecular segment names to read from either MOL2 or PSF/CRD file. If this parameter is defined, then any segment/chains not listed are not loaded. This provides a convenient way to remove cofactors, counterions and solvent without modifying the original file.	Comma separated list of segment name strings (without any spaces)	Empty (i.e. all segments read from file)	Comma separated list of segment name strings
`RECEPTOR_FLEX`	Define terminal OH and NH3+ groups withing this distance of docking volume as flexible.	float (Angstroms)	Undefined (rigid receptor)	>0.0 (3.0 is a reasonable value)
Advanced parameters (should not need to be changed by the user)				
`RECEPTOR_ALL_H`	Disable the removal of explicit non-polar hydrogens from the receptor model. *Not recommended*	boolean	FALSE	TRUE or FALSE
`DIHEDRAL_STEP`	Maximum mutation step size for receptor dihedral degrees of freedom.	float (degrees)	30.0	>0.0

3.8.2 Ligand definition

Ligand definition parameters need only be defined if you wish to introduce tethering of some or all of the ligand degrees of freedom. If you are running conventional free docking then this section is not required. All ligand definition parameters should be defined in `SECTION LIGAND`. Note that the ligand input SD file continues to be specified directly on the `rbdock` command-line and not in the system definition file.

Table 3.9: Ligand definition parameters

Parameter	Description	Type	Default	Range of values
Main user parameters				
`TRANS_MODE`	Sampling mode for ligand translational degrees of freedom	enumerated string literal	`FREE`	`FIXED`, `TETHERED`, `FREE`
`ROT_MODE`	Sampling mode for ligand whole-body rotational degrees of freedom	enumerated string literal	`FREE`	`FIXED`, `TETHERED`, `FREE`
`DIHEDRAL_MODE`	Sampling mode for ligand internal dihedral degrees of freedom	enumerated string literal	`FREE`	`FIXED`, `TETHERED`, `FREE`
`MAX_TRANS`	(for `TRANS_MODE = TETHERED` only) Maximum deviation allowed from reference centre of mass	float (Angstroms)	1.0	>0.0
`MAX_ROT`	(for `ROT_MODE = TETHERED` only) Maximum deviation allowed from orientation for reference principle axes	float (degrees)	30.0	>0.0–180.0
`MAX_DIHEDRAL`	(for `DIHEDRAL_MODE = TETHERED` only) Maximum deviation allowed from reference dihedral angles for any rotatable bond	float (degrees)	30.0	>0.0–180.0
Advanced parameters (should not need to be changed by the user)				
`TRANS_STEP`	Maximum mutation step size for ligand translational degrees of freedom	float (Angstroms)	2.0	>0.0
`ROT_STEP`	Maximum mutation step size for ligand whole-body rotational degrees of freedom	float (degrees)	30.0	>0.0
`DIHEDRAL_STEP`	Maximum mutation step size for ligand internal dihedral degrees of freedom	float (degrees)	30.0	>0.0

3.8.3 Solvent definition

Solvent definition parameters need only be defined if you wish to introduce explicit structural waters into the docking calculation, otherwise this section is not required. All solvent definition parameters should be defined in `SECTION SOLVENT`.

Table 3.10: Solvent definition parameters

Parameter	Description	Type	Default	Range of values
Main user parameters				
FILE	Name of explicit solvent PDB file	File name string	No default value (mandatory parameter)	Valid PDB filename
TRANS_MODE	Sampling mode for solvent translational degrees of freedom. If defined here, the value overrides the per-solvent translational sampling modes defined in the solvent PDB file	enumerated string literal	FREE	FIXED, TETHERED, FREE
ROT_MODE	Sampling mode for solvent whole-body rotational degrees of freedom. If defined here, the value overrides the per-solvent rotational sampling modes defined in the solvent PDB file	enumerated string literal	FREE	FIXED, TETHERED, FREE
MAX_TRANS	(for TRANS_MODE = TETHERED waters only) Maximum deviation allowed from reference water oxygen positions. The same value is applied to all waters with TRANS_MODE = TETHERED; it is no possible currently to define per-solvent MAX_TRANS values	float (Angstroms)	1.0	>0.0
MAX_ROT	(for ROT_MODE = TETHERED waters only) Maximum deviation allowed from orientation of reference principal axes. The same value is applied to all waters with ROT_MODE = TETHERED; it is no possible currently to define per-solvent MAX_ROT values	float (degrees)	30.0	>0.0–180.0
OCCUPANCY	Controls occupancy state sampling for all explicit solvent. If defined here, the values overrides the per-solvent occupancy states defined in the solvent PDB file	float	1.0	0.0–1.0
Advanced parameters (should not need to be changed by the user)				
TRANS_STEP	Maximum mutation step size for solvent translational degrees of freedom	float (Angstroms)	2.0	>0.0
ROT_STEP	Maximum mutation step size for solvent wholebody rotational degrees of freedom	float (degrees)	30.0	>0.0
OCCUPANCY_STEP	Maximum mutation step size for solvent occupancy state degrees of freedom	float (degrees)	1.0	0.0–1.0

Solvent occupancy state sampling OCCUPANCY = 0 permanently disables all solvent; OCCUPANCY = 1.0 permanently enables all solvent; OCCUPANCY between 0 and 1 activates variable occupancy state sampling, where the value represents the initial probability that the solvent molecule is enabled. For example, OCCUPANCY = 0.5 means that the solvent is enabled in 50 % of the initial GA population. However, the probability that the solvent is actually enabled in the final docking solution will depend on the particular ligand, the scoring function terms, and on the penalty for solvent binding. The occupancy state chromosome value is managed as a continuous variable between 0.0 and 1.0, with a nominal mutation step size of 1.0. Chromosome values lower than the occupancy threshold (defined as 1.0 - OCCUPANCY) result in the solvent being disabled; values higher than the threshold result in the solvent being enabled.

3.8.4 Cavity mapping

The cavity mapping section is mandatory. You should choose one of the mapping algorithms shown below. All mapping parameters should be defined in `SECTION MAPPER`.

Table 3.11: Two sphere site mapping parameters

Parameter	Description	Type	Default	Range of values
Main user parameters				
`SITE_MAPPER`	Mapping algorithm specifier	string literal	`RbtSphereSiteMapper`	fixed
`CENTER`	(x,y,z) center of cavity mapping region	Bracketed cartesian coordinate string (x,y,z)	None	None
`RADIUS`	Radius of cavity mapping region	float (Angstroms)	10.0	>0.0 (10.0–20.0 suggested range)
`SMALL_SPHERE`	Radius of small probe	float (Angstroms)	1.5	>0.0 (1.0–2.0 suggested range)
`LARGE_SPHERE`	Radius of large probe	float (Angstroms)	4.0	>`SMALL_SPHERE` (3.5–6.0 suggested range)
`MAX_CAVITIES`	Maximum number of cavities to accept (in descending order of size)	integer	99	>0
Advanced parameters (less frequently changed by the user)				
`VOL_INCR`	Receptor atom radius increment for excluded volume	float (Angstroms)	0.0	>=0.0
`GRID_STEP`	Grid resolution for mapping	float (Angstroms)	0.5	>0.0 (0.3–0.8 suggested range)
`MIN_VOLUME`	Minimum cavity volume to accept (in $Å^3$, not grid points)	float ($Angstroms^3$)	100	>0 (100–300 suggested range)

Table 3.12: Reference ligand site mapping parameters

Parameter	Description	Type	Default	Range of values
Main user parameters				
SITE_MAPPER	Mapping algorithm specifier	string literal	RbtLigandSiteMapper	fixed
REF_MOL	Reference ligand SD file name	string	ref.sd	None
RADIUS	Radius of cavity mapping region	float (Angstroms)	10.0	>0.0 (10.0–20.0 suggested range)
SMALL_SPHERE	Radius of small probe	float (Angstroms)	1.5	>0.0 (1.0–2.0 suggested range)
LARGE_SPHERE	Radius of large probe	float (Angstroms)	4.0	>SMALL_SPHERE (3.5–6.0 suggested range)
MAX_CAVITIES	Maximum number of cavities to accept (in descending order of size)	integer	99	>0
Advanced parameters (less frequently changed by the user)				
VOL_INCR	Receptor atom radius increment for excluded volume	float (Angstroms)	0.0	>=0.0
GRID_STEP	Grid resolution for mapping	float (Angstroms)	0.5	>0.0 (0.3–0.8 suggested range)
MIN_VOLUME	Minimum cavity volume to accept (in $Å^3$, not grid points)	float ($Å^3$)	100	>0 (100–300 suggested range)

3.8.5 Cavity restraint

The cavity restraint penalty function is mandatory and is designed to prevent the ligand from exiting the docking site. The function is calculated over all non-hydrogen atoms in the ligand (and over all explicit water oxygens that can translate). The distance from each atom to the nearest cavity grid point is calculated. If the distance exceeds the value of RMAX, a penalty is imposed based on the value of (distance - RMAX). The penalty can be either linear or quadratic depending on the value of the QUADRATIC parameter. It should not be necessary to change any the parameters in this section. Note that the docking protocol itself will manipulate the WEIGHT parameter, so any changes made to WEIGHT will have no effect.

```
SECTION CAVITY
   SCORING_FUNCTION RbtCavityGridSF
   WEIGHT 1.0
   RMAX 0.1
   QUADRATIC FALSE
END_SECTION
```

3.8.6 Pharmacophore restraints

This section need only be defined if you wish to dock with pharmacophore restraints. If you are running conventional free docking then this section is not required. All pharmacophore definition parameters should be defined in SECTION PHARMA.

Table 3.13: Pharmacophore restraint parameters

Parameter	Description	Type	Default	Range of values
CONSTRAINT_FILE	Mandatory pharmacophore restraints file	File name string	None (mandatory parameter)	Valid file name
OPTIONAL_FILE	Optional pharmacophore restraints file	File name string	None (optional parameter)	Valid file name, or empty
NOPT	Number of optional restraints that should be met	Integer	0	Between 0 and number of restraints in OPTIONAL_FILE
WRITE_ERRORS	Ligands with insufficient pharmacophore features to match the mandatory restraints are always removed prior to docking. If this parameter is true, the pre-filtered ligands are written to an error SD file with the same root name as the docked pose output SD file, but with an _errors.sd suffix. If false, the pre-filtered ligands are not written.	Boolean	FALSE	TRUE of FALSE
WEIGHT	Overall weight for the pharmacophore penalty function	Float	1.0	>=0.0

Calculation of mandatory restraint penalty The list of ligand atoms that matches each restraint type in the mandatory restraints file is precalculated for each ligand as it is loaded. If the ligand contains insufficient features to satisfy all of the mandatory restraints the ligand is rejected and is not docked. Note that the rejection is based purely on feature counts and does not take into account the possible geometric arrangements of the features. Rejected ligands are optionally written to an error SD file. The penalty for each restraint is based on the distance from the nearest matching ligand atom to the pharmacophore restraint centre. If the distance is less than the defined tolerance (restraint sphere radius), the penalty is zero. If the distance is greater than the defined tolerance a quadratic penalty is applied, equal to (nearest distance - tolerance)^2.

Calculation of optional restraint penalty The individual restraint penalties for each restraint in the optional restraints file are calculated in the same way as for the mandatory penalties. However, only the NOPT lowest scoring (least penalised) restraints are summed for any given docking pose. Any remaining higher scoring optional restraints are ignored and do not contribute to the total pharmacophore restraint penalty.

Calculation of overall restraint penalty The overall pharmacophore restraint penalty is the sum of the mandatory restraint penalties and the NOPT lowest scoring optional restraint penalties, multiplied by the WEIGHT parameter value.

3.8.7 NMR restraints

To be completed. However, this feature has rarely been used.

3.8.8 Example system definition files

Full system definition file with all sections and common parameters enumerated explicitly:

```
RBT PARAMETER_FILE_V1.00
TITLE HSP90-PU3-lig-cavity, solvent flex=5
RECEPTOR_FILE PROT_W3_flex.mol2
RECEPTOR_SEGMENT_NAME PROT
RECEPTOR_FLEX 3.0
SECTION SOLVENT
   FILE PROT_W3_flex_5.pdb
   TRANS_MODE TETHERED
   ROT_MODE TETHERED
   MAX_TRANS 1.0
   MAX_ROT 30.0
   OCCUPANCY 0.5
END_SECTION
SECTION_LIGAND
   TRANS_MODE FREE
   ROT_MODE FREE
   DIHEDRAL_MODE FREE
   MAX_TRANS 1.0
   MAX_ROT 30.0
   MAX_DIHEDRAL 30.0
END_SECTION
SECTION MAPPER
   SITE_MAPPER RbtLigandSiteMapper
   REF_MOL ref.sd
   RADIUS 5.0
   SMALL_SPHERE 1.0
   MIN_VOLUME 100
   MAX_CAVITIES 1
   VOL_INCR 0.0
   GRIDSTEP 0.5
END_SECTION
SECTION CAVITY
   SCORING_FUNCTION RbtCavityGridSF
   WEIGHT 1.0
END_SECTION
SECTION PHARMA
   SCORING_FUNCTION RbtPharmaSF
   WEIGHT 1.0
   CONSTRAINTS_FILE mandatory.const
   OPTIONAL FILE optional.const
   NOPT 3
   WRITE_ERRORS TRUE
END_SECTION
```

3.9 Molecular files and atom typing

Macromolecular targets (protein or RNA) are input from Tripos MOL2 files (`RbtMOL2FileSource` class) or from pairs of Charmm PSF (`RbtPsfFileSource` class) and CRD (`RbtCrdFileSource` class) files. Ligands are input from MDL Information Systems (MDL) structure data (SD) files (`RbtMdlFileSource` class). Explicit structural waters are input optionally from PDB files (`RbtPdbFileSource` class). Ligand docking poses are output to MDL SD files.

The RxDock scoring functions have been defined and validated for implicit non-polar hydrogen (extended carbon) models only. If you provide all-atom models, be aware that the non-polar hydrogens will be removed automatically. Polar hydrogens must be defined explicitly in the molecular files, and are not added by RxDock. Positive ionisable and negative ionisable groups can be automatically protonated and deprotonated respectively to create common charged groups such as guanidinium and carboxylic acid groups.

MOL2 is now the preferred file format for RxDock as it eliminates many of the atom typing issues inherent in preparing and loading PSF files. The use of PSF/CRD files is strongly discouraged. The recommendation is to prepare an all-atom MOL2 file with correct Tripos atom types assigned, and allow RxDock to remove non-polar hydrogens on-the-fly.

3.9.1 Atomic properties

RxDock requires the following properties to be defined per atom. Depending on the file format, these properties may be loaded directly from the molecular input file, or derived internally once the model is loaded:

- Cartesian (x,y,z) coordinates
- Element (atomic number)
- Formal hybridisation state (sp, sp2, sp3, aromatic, trigonal planar)
- Formal charge
- Distributed formal charge (known informally as group charge)
- Tripos force field type (RxDock uses a modified version of the Sybyl 5.2 types, extended to include carbon types with implicit non-polar hydrogens)
- Atom name
- Substructure (residue) name
- Atomic radius (assigned per element from `$RBT_ROOT/data/RbtElements.dat`)

Note: The RxDock scoring functions do not use partial charges and therefore partial charges do not have to be defined. The atomic radii are simplified radii defined per element, and are used for cavity mapping and in the polar scoring function term, but are not used in the vdW scoring function term. The latter has its own indepedent parameterisation based on the Tripos force field types.

3.9.2 Difference between formal charge and distributed formal charge

The formal charge on an atom is always an integer. For example, a charged carboxylic acid group (COO-) can be defined formally as a formal double bond to a neutral oxygen sp2, and a formal single bond to a formally charged oxygen sp3. In reality of course, both oxygens are equivalent. RxDock distributes the integer formal charge across all equivalent atoms in the charged group that are topologically equivalent. In negatively charged acid groups, the formal charge is distributed equally between the acid oxygens. In positively charged amines, the formal charge is distributed equally between the hydrogens. In charged guanidinium, amidinium, and imidazole groups, the central carbon also receives an equal portion of the formal charge (in addition to the hydrogens). The distributed formal charge is also known as the group charge. The polar scoring functions in RxDock use the distributed formal charge to scale the polar interaction strength of the polar interactions.

3.9.3 Parsing a MOL2 file

`MOLECULE`, `ATOM`, `BOND`, and `SUBSTRUCTURE` records are parsed. The atom name, substructure name, Cartesian coordinates and Tripos atom type are read directly for each atom. The element type (atomic number) and formal hybridisation state are derived from the Tripos type using an internal lookup table. Formal charges are not read from the MOL2 file and do not have to be assigned correctly in the file. Distributed formal charges are assigned directly by RxDock based on standard substructure and atom names as described below.

3.9.4 Parsing an SD file

Cartesian coordinates, element and formal charge are read directly for each atom. Formal bond orders are read for each bond. Atom names are derived from element name and atom ID (e.g. C1, N2, C3). The substructure name is `MOL`. Formal hybridisation states are derived internally for each atom based on connectivity patterns and formal bond orders. The Tripos types are asssigned using internal rules based on atomic number, formal hybridisation state and formal charges. The integer formal charges are distributed automatically across all topologically equivalent atoms in the charged group.

3.9.5 Assigning distributed formal charges to the receptor

RxDock provides a file format independent method for assigning distributed formal charges directly to the receptor atoms, which is used by the MOL2 and PSF/CRD file readers. The method uses a lookup table based on standard substructure and atom names, and does not require the integer formal charges to be assigned to operate correctly.

The lookup table file is `$RBT_ROOT/data/sf/RbtIonicAtoms.prm`. Each section name represents a substructure name that contains formally charged atoms. The entries within the section represent the atom names and distributed formal charges for that substructure name. The file provided with RxDock contains entries for all standard amino acids and nucleic acids, common metals, and specific entries required for processing the GOLD CCDC/Astex validation sets.

Important: You may have to extend `RbtIonicAtoms.prm` if you are working with non-standard receptor substructure names and/or atom names, in order for the distributed formal charges to be assigned correctly.

3.10 File formats

3.10.1 .prm file format

The .prm file format is an RxDock-specific text format and is used for:

- system definition files (known previously as receptor `.prm` files)
- scoring function definition files
- search protocol definition files

The format is simple and allows for an arbitrary number of named parameter/value pairs to be defined, optionally divided into named sections. Sections provide a namespace for parameter names, to allow parameter names to be duplicated within different sections. The key features of the format are:

- The first line of the file must be `RBT_PARAMETER_FILE_V1.00` with no preceeding whitespace.
- Subsequent lines may contain either:
  1. comment lines
  2. reserved keywords `TITLE`, `SECTION`, or `END_SECTION`
  3. parameter name/value pairs
- Comment lines should start with a `#` character in the first column with no preceeding whitespace, and are ignored.
- The reserved words must start in the first column with no preceeding whitespace.
- The `TITLE` record should occur only once in the file and is used to provide a title string for display by various scripts such as run `rbscreen.pl`. The keyword should be followed by a single space character and then the title string, which may contain spaces. If the `TITLE` line occurs more than once, the last occurence is used.
- `SECTION` records can occur more than once, and should always be paired with a closing `END_SECTION` record. The keyword should be followed by a single space character and then the section name, which may NOT itself contain spaces. All section names must be unique within a .prm file. All parameter name/value pairs within the `SECTION` / `END_SECTION` block belong to that section.
- Parameter name/value pairs are read as free-format tokenised text and can have preceeding, trailing, and be separated by arbitrary whitespace. This implies that the parameter name and value strings themselves are not allowed to contain any spaces. The value strings are interpreted as numeric, string, or boolean values as appropriate for that parameter. Boolean values should be entered as `TRUE` or `FALSE` uppercase strings.

Caution: The current implementation of the .prm file reader does not tolerate a tab character immediately following the `TITLE` and `SECTION` keywords. It is very important that the first character after the `SECTION` keyword in particular is a true space character, otherwise the reserved word will not be detected and the parameters for that section will be ignored.

Example .prm file In the following example, `RECEPTOR_FILE` is defined in the top level namespace. The remaining parameters are defined in the `MAPPER` and `CAVITY` namespaces. The indentation is for readability, and has no significance in the format.

```
RBT_PARAMETER_FILE_V1.00
TITLE 4dfr oxido-reductase

RECEPTOR_FILE 4dfr.mol2
```

(continues on next page)

(continued from previous page)

```
SECTION MAPPER
   SITE_MAPPER RbtLigandSiteMapper
   REF_MOL 4dfr_c.sd
   RADIUS 6.0
   SMALL_SPHERE 1.0
   MIN_VOLUME 100
   MAX_CAVITIES 1
   VOL_INCR 0.0
   GRIDSTEP 0.5
END_SECTION

SECTION CAVITY
   SCORING_FUNCTION RbtCavityGridSF
   WEIGHT 1.0
END_SECTION
```

3.10.2 Water PDB file format

RxDock requires explicit water PDB files to be in the style as output by the Dowser program. In particular:

- Records can be `HETATM` or `ATOM`
- The atom names must be `OW`, `H1` and `H2`
- The atom records for each water molecule must belong to the same subunit ID
- The subunit IDs for different waters must be distinct, but do not have to be consecutive
- The atom IDs are not used and do not have to be consecutive (they can even be duplicated)
- The order of the atom records within a subunit is unimportant
- The temperature factor field of the water oxygens can be used to define the per-solvent flexibility modes. The temperature factors of the water hydrogens are not used.

Table 3.14: Conversion of temperature values to solvent flexibility modes

PDB temperature factor	Solvent translational flexibility	Solvent rotational flexibility
0	FIXED	FIXED
1	FIXED	TETHERED
2	FIXED	FREE
3	TETHERED	FIXED
4	TETHERED	TETHERED
5	TETHERED	FREE
6	FREE	FIXED
7	FREE	TETHERED
8	FREE	FREE

Example Valid RxDock PDB file for explicit, flexible waters:

```
REMARK tmp 1YET.pdb xtal_hoh.pdb
HETATM 3540 OW HOH W 106 28.929 12.684 20.864 1.00 1.0
HETATM 3540 H1 HOH W 106 28.034 12.390 21.200 1.00
HETATM 3540 H2 HOH W 106 29.139 12.204 20.012 1.00
HETATM 3542 OW HOH W 108 27.127 14.068 22.571 1.00 2.0
```

(continues on next page)

(continued from previous page)

```
HETATM 3542 H1 HOH W 108 26.632 13.344 23.052 1.00
HETATM 3542 H2 HOH W 108 27.636 13.673 21.806 1.00
HETATM 3679 OW HOH W 245 27.208 10.345 27.250 1.00 3.0
HETATM 3679 H1 HOH W 245 27.657 10.045 26.409 1.00
HETATM 3679 H2 HOH W 245 26.296 10.693 27.036 1.00
HETATM 3680 OW HOH W 246 31.737 12.425 21.110 1.00 4.0
HETATM 3680 H1 HOH W 246 31.831 12.448 22.106 1.00
HETATM 3680 H2 HOH W 246 30.775 12.535 20.863 1.00
```

3.10.3 Pharmacophore restraints file format

Pharmacophore restraints are defined in a simple text file, with one restraint per line. Each line should contain the following values, separated by commas or whitespace:

```
x y z coords of restraint centre, tolerance (in Angstroms), restraint type string
```

The supported restraint types are:

Table 3.15: Pharmacophore restraint types

String	Description	Matches
`Any`	Any atom	Any non-hydrogen atom
`Don`	H-bond donor	Any neutral donor hydrogen
`Acc`	H-bond acceptor	Any neutral acceptor
`Aro`	Aromatic	Any aromatic ring centre (pseudo atom)
`Hyd`	Hydrophobic	Any non-polar hydrogens (if present), any C sp3 or S sp3, any C or S not bonded to O sp2, any Cl, Br, I
`Hal`	Hydrophobic, aliphatic	Subset of Hyd, sp3 atoms only
`Har`	Hydrophobic, aromatic	Subset of Hyd, aromatic atoms only
`Ani`	Anionic	Any atom with negative distributed formal charge
`Cat`	Cationic	Any atom with positive distributed formal charge

3.11 Programs

Programs summary tables:

Table 3.16: Core RxDock C++ executables

Executable	Used for	Description
`rbcavity`	Preparation	Cavity mapping and preparation of docking site (.as) file.
`rbcalcgrid`	Preparation	Calculation of vdW grid files (usually called by `make_grid.csh` wrapper script).
`rbdock`	Docking	The main RxDock docking engine itself.

Table 3.17: Auxiliary RxDock programs

Executable	Used for	Description
`sdtether`	Preparation	Prepares a ligand SD file for tethered scaffold docking. Annotates ligand SD file with tethered substructure atom indices. Requires Open Babel Python bindings.
`rbhtfinder`	Preparation	Used to optimise a high-throughput docking protocol from an initial exhaustive docking of a small representative ligand library. Parametrize a multi-step protocol for your system.
`make_grid.csh`	Preparation	Creates the vdW grid files required for grid-based docking protocols (`dock_grid.prm` and `dock_solv_grid.prm`). Simple front-end to `rbcalcgrid`.
`rbconvgrid`	Analysis	Converts RxDock vdW grids to InsightII grid format for visualisation.
`rbmoegrid`	Analysis	Converts RxDock vdW grids to MOE grid format for visualisation.
`rblist`	Analysis	Outputs miscellaneous information for ligand SD file records.
`sdrmsd`	Analysis	Calculation of ligand Root Mean Squared Displacement (RMSD) between reference and docked poses, taking into account ligand topological symmetry. Requires Open Babel Python bindings.
`sdfilter`	Analysis	Utility for filtering SD files by arbitrary data field expressions. Useful for simple post-docking filtering by score components.
`sdsort`	Analysis	Utility for sorting SD files by arbitrary data field. Useful for simple post-docking filtering by score components.
`sdreport`	Analysis	Utility for reporting SD file data field values. Output in tab-delimited or CSV format.
`sdsplit`	Utility	Splits an SD file into multiple smaller SD files of fixed number of records.
`sdmodify`	Utility	Sets the molecule title line of each SD record equal to a given SD data field.

3.11.1 Programs reference

rbdock

`rbdock` – The RxDock docking engine itself.

```
$RBT_ROOT/bin/rbdock
{-i input ligand MDL SD file}
{-o output MDL SD file}
{-r system definition.prm file}
{-p docking protocol.prm file}
[-n number of docking runs/ligand]
[-s random seed]
[-T debug trace level]
[[-t SCORE.INTER threshold] | [-t filter definition file]]
[-ap -an -allH -cont]
```

Simple exhaustive docking

The minimum requirement for rbdock is to specify the input (`-i`) and output (`-o`) ligand SD file names, the system definition .prm file (`-r`) and the docking protocol .prm file (`-p`). This will perform one docking run per ligand record in the input SD file and output all docked ligand poses to the output SD file. Use `-n` to increase the number of docking runs per ligand record.

High-throughput docking, option 1

The `-t` and `-cont` options can be used to construct high-throughput protocols. If the argument following `-t` is numeric it is interpreted as a threshold value for `SCORE.INTER`, the total intermolecular score between ligand and receptor/solvent. In the absence of -cont, the threshold acts as an early termination filter, and the docking runs for each ligand will be terminated early once the threshold value has been exceeded. Note that the threshold is applied only at the end of each individual docking run, not during the runs themselves. If the `-cont` (continue) option is specified as well, the threshold acts as an output pose filter instead of a termination filter. The docking runs for each ligand run to completion as in the exhaustive case, but only the docking poses that exceed the threshold value of `SCORE.INTER` are written to the output SD file.

High throughput docking, option 2

Alternatively, if the argument following `-t` is non-numeric it is interpreted as a filter definition file. The filter definition file can be used to define multiple termination filters and multiple output pose filters in a generic way. Any docking score component can be used in the filter definitions. `run_rbscreen.pl` generates a filter definition file for multi-stage, high-throughput docking, with progressive score thresholds for early termination of poorly performing ligands. The use of filter definition files is preferred over the more limited `SCORE.INTER` filtering described above, whose use is now deprecated.

Automated ligand protonation/deprotonation

The `-ap` option activates the automated protonation of ligand positive ionisable centres, notably amines, guanidines, imidazoles, and amidines. The `-an` option activates the automated deprotonation of ligand negative ionisable centres, notably carboxylic acids, phosphates, phosphonates, sulphates, and sulphonates. The precise rules used by RxDock for protonation and deprotonation are quite crude, and are not user-customisable. Therefore these flags are not recommended for detailed validation experiments, in which care should be taken that the ligand protonation states are set correctly in the input SD file. Note that RxDock is not capable of converting ionised centres back to the neutral form; these are unidirectional transformations.

Control of ligand non-polar hydrogens

By default, RxDock uses an implicit non-polar hydrogen model for receptor and ligand, and all of the scoring function validation has been performed on this basis. If the `-allH` option is not defined (recommended), all explicit non-polar hydrogens encountered in the ligand input SD file are removed, and only the polar hydrogens (bonded to O, N, or S) are retained. If the `-allH` option is defined (not recommended), no hydrogens are removed from the ligand. Note that RxDock is not capable of adding explicit non-polar hydrogens, if none exist. In other words, the `-allH` option disables hydrogen removal, it does not activate hydrogen addition. You should always make sure that polar hydrogens are defined explicitly. If the ligand input SD file contains no explicit non-polar hydrogens, the `-allH` option has no effect. Receptor protonation is controlled by the system definition prm file.

rbcavity

`rbcavity` – Cavity mapping and preparation of docking site (`.as`) file.

```
$RBT_ROOT/bin/rbcavity
{-r system definition .prm file}
[-ras -was -d -v -s]
[-l distance from cavity]
[-b border]
```

Exploration of cavity mapping parameters

```
rbcavity -r .prm file
```

You can run rbcavity with just the `-r` argument when first preparing a new receptor for docking. This allows you to explore rapidly the impact of the cavity mapping parameters on the generated cavities, whilst avoiding the overhead of actually writing the docking site (`.as`) file to disk. The number of cavities and volume of each cavity are written to standard output.

Visualisation of cavities

```
rbcavity -r .prm file -d
```

If you have access to InsightII you can use the `-d` option to dump the cavity volumes in InsightII grid file format. There is no need to write the docking site (`.as`) file first. The InsightII grid files should be loaded into the reference coordinate space of the receptor and contoured at a contour level of 0.99.

Writing the docking site (.as) file

```
rbcavity -r .prm file -was
```

When you are happy the mapping parameters, use the `-was` option to write the docking site (`.as`) file to disk. The docking site file is a binary file that contains the cavity volumes in a compact format, and a pre-calculated cuboid grid extending over the cavities. The grid represents the distance from each point in space to the nearest cavity grid point, and is used by the cavity penalty scoring function. Calculating the distance grid can take a long time (whereas the cavity mapping itself is usually very fast), hence the `-was` option should be used sparingly.

Analysis of cavity atoms

```
rbcavity -r .prm file -ras -l distance
```

Use the `-l` options to list the receptor atoms within a given distance of any of the cavity volumes, for example to determine which receptor OH/NH3+ groups should be flexible. This option requires access to the pre-calculated distance grid embedded within the docking site (.as) file, and is best used in combination with the `-ras` option, which loads a previously generated docking site file. This avoids the time consuming step of generating the cavity distance grid again. If `-l` is used without `-ras`, the cavity distance grid will be calculated on-the-fly each time.

Miscellaneous options

The `-s` option writes out various statistics on the cavity and on the receptor atoms in the vicinity of the cavity. These values have been used in genetic programming model building for docking pose false positive removal. The `-v` option writes out the receptor coordinates in PSF/CRD format for use by the rDock Viewer (not documented here). Note that the PSF/CRD files are not suitable for simulation purposes, only for visualisation, as the atom types are not set correctly. The `-b` option controls the size of the cavity distance grid, and represents the border beyond the actual cavity volumes. It should not be necessary to vary this parameter (default = 8 Å) unless longer-range scoring functions are implemented.

rbcalcgrid

`rbcalcgrid` – Calculation of vdW grid files (usually called by `make_grid.csh` wrapper script).

```
$RBT_ROOT/bin/rbcalcgrid
{-r system definition file}
{-o output suffix for generated grids}
{-p vdW scoring function prm file}
[-g grid step]
[-b border]
```

Note that, unlike `rbdock` and `rbcavity`, spaces are not tolerated between the command-line options and their corresponding arguments. See `$RBT_ROOT/bin/make_grid.csh` for common usage.

make_grid.csh

Creates vdW grids for all receptor `.prm` files listed on command line. Front-end to `rbcalcgrid`.

rbconvgrid

rbmoegrid

`rbmoegrid` – Calculates grids for a given atom type.

```
rbmoegrid -o <OutputRoot> -r <ReceptorPrmFile> -p <SFPrmFile> [-g <GridStep> -b
↪<border> -t <tripos_type>]

-o <OutFileName> (.grd is suffiexed)
-r <ReceptorPrmFile> - receptor param file (contains active site params)
-p <SFPrmFile> - scoring function param file (default calcgrid_vdw.prm)
-g <GridStep> - grid step (default = 0.5A)
-b <Border> - grid border around docking site (default = 1.0A)
-t <AtomType> - Tripos atom type (default is C.3)
```

sdrmsd

`sdrmsd` – calculation of ligand root mean squared displacement (RMSD) between reference and docked poses. It takes into account molecule topological symmetry. Requires Open Babel Python bindings.

```
$RBT_ROOT/bin/sdrmsd [options] {reference SD file} {input SD file}
```

With two arguments

`sdrmsd` calculates the RMSD between each record in the input SD file and the first record of the reference SD file. If there is a mismatch in the number of atoms, the record is skipped and the RMSD is not calculated. The RMSD is calculated over the heavy (non-hydrogen) atoms only. Results are output to standard output. If some record was skipped, a warning message will be printed to standard error.

With fitting

A molecular superposition will be done before calculation of the RMSD. The output will specify an RMSD FIT calculation was done.

```
sdrmsd -o output.sdf reference.sdf input.sdf
sdrmsd --out=output.sdf reference.sdf input.sdf
```

Output a SD file

This option will write an output SD file with the input molecules adding an extra RMSD field to the file. If fitting was done, the molecule coordinates will also be fitted to the reference.

```
sdrmsd -o output.sdf reference.sdf input.sdf
sdrmsd --out=output.sdf reference.sdf input.sdf
```

sdtether

`sdtether` – Prepares a ligand SD file for tethered scaffold docking. Requires Open Babel Python bindings. Annotates ligand SD file with tethered substructure atom indices.

```
$RBT_ROOT/bin/sdtether {ref. SDfile} {in SDfile} {out SDfile} "{SMARTS query}"
```

`sdtether` performs the following actions:

- Runs the SMARTS query against the reference SD file to determine the tethered substructure atom indices and coordinates.
- If more than one substructure match is retrieved (e.g. due to topological symmetry, or if the query is too simple) all substructure matchs are retained as the reference and all ligands will be tethered according to all possible matches.
- Runs the SMARTS query against each record of the input ligand SD file in turn.
- For each substructure match, the ligand coordinates are transformed such that the principal axes of the matching substructure coordinates are aligned with the reference substructure coordinates.
- In addition, an SD data field is added to the ligand record which lists the atom indices of the substructure match, for later retrieval by RxDock.

- Each transformed ligand is written to the output SD file.
- Note that if the SMARTS query returns more than one substructure match for a ligand, that ligand is written multiple times to the output file, once for each match, each of which will be docked independently with different tethering information.

sdfilter

`sdfilter` – Post-process an SD file by filtering the records according to data fields or attributes.

```
sdfilter -f '$<DataField> <Operator> <Value>' [-s <DataField>] [sdFiles]
```

or

```
sdfilter -f <filename> [-s <DataField>] [sdFiles]
```

Note: Multiple filters are allowed and are OR'd together. Filters can be provided in a file, one per line. Standard Perl operators should be used. e.g.

```
eq ne lt gt le ge # for strings
== != < > <= >=   # for numeric
```

`_REC` (record #) is provided as a pseudo-data field. If `-s` option is used, `_COUNT` (#occurrences of DataField) is provided as a pseudo-data field. If SD file list not given, reads from standard input. Output is to standard output.

For example, if `results.sd` constains multiple ligands each having multiple poses (ordered by score), then running

```
sdfilter -f'$_COUNT == 1' results.sd
```

will get you the first entry for each ligand.

sdreport

`sdreport` – Produces text summaries of SD records.

```
sdreport [-l] [-t [<FieldName, FieldName...>]] [-c <FieldName, FieldName...>] [-id
↪<IDField>] [-nh] [-o] [-s] [-sup] [sdFiles]

-l (list format) output all data fields for each record as processed
-t (tab format) tabulate selected fields for each record as processed
-c (csv format) comma delimited output of selected fields for each record as processed
-s (summary format) output summary statistics for each unique value of ligand ID
-sup (supplier format) tabulate supplier details (from Catalyst)
-id <IDField> data field to use as ligand ID
-nh don't output column headings in -t and -c formats
-o use old (v3.00) score field names as default columns in -t and -c formats, else␣
↪use v4.00 field names
-norm use normalised score filed names as default columns in -t and -c formats␣
↪(normalised = score / #ligand heavy atoms)
```

Note: If `-l`, `-t` or `-c` are combined with `-s`, the listing/table is output withing each ligand summary. `-sup` should not be combined with other options. Default field names for `-t` and `-c` are RiboDock score field names. Default ID field name is `Name`. If `sdFiles` not given, reads from standard input. Output is to standard output.

---

sdsplit

`sdsplit` – Splits SD records into multiple files of equal size.

```
sdsplit [-<RecSize>] [-o <OutputRoot>] [sdFiles]

-<RecSize> record size to split into (default = 1000 records)
-o <OutputRoot> Root name for output files (default = tmp)
```

Note: If SD file list not given, reads from standard input.

sdsort

`sdsort` – Sorts SD records by given data field.

```
sdsort [-n] [-r] [-f <DataField>] [sdFiles]

-n numeric sort (default is text sort)
-r descending sort (default is ascending sort)
-f <DataField> specifies sort field
-s fast mode. Sorts the records for each named compound independently (must be␣
↪consecutive)
-id <NameField> specifies compound name field (default = 1st title line)
```

Note: `_REC` (record #) is provided as a pseudo-data field. If SD file list not given, reads from standard input. Output is to standard output. Fast mode can be safely used for partial sorting of huge SD files of raw docking hits without running into memory problems.

sdmodify

`sdmodify` – Script to set the first title line equal to a given data field.

```
sdmodify -f <DataField> [sdFiles]
```

Note: If `sdFiles` not given, reads from standard input. Output is to standard output.

rbhtfinder

`rbhtfinder` – Script that simulates the result of a high throughput protocol.

```
1st) exhaustive docking of a small representative part of the
     whole library.
2nd) Store the result of sdreport -t over that exhaustive dock.
     in file  that will be the input of this
     script.
3rd) rbhtfinder <sdreport_file> <output_file> <thr1max> <thr1min> <ns1> <ns2>
     <ns1> and <ns2> are the number of steps in stage 1 and in
```

(continues on next page)

(continued from previous page)

```
    stage 2. If not present, the default values are 5 and 15
    <thrmax> and <thrmin> setup the range of thresholds that will
    be simulated in stage 1. The threshold of stage 2 depends
    on the value of the threshold of stage 1.
    An input of -22 -24 will try protocols:
            5   -22     15      -27
            5   -22     15      -28
            5   -22     15      -29
            5   -23     15      -28
            5   -23     15      -29
            5   -23     15      -30
            5   -24     15      -29
            5   -24     15      -30
            5   -24     15      -31
    Output of the program is a 7 column values. First column
    represents the time. This is a percentage of the time it
    would take to do the docking in exhaustive mode, i.e.
    docking each ligand 100 times. Anything
    above 12 is too long.
    Second column is the first percentage. Percentage of
    ligands that pass the first stage.
    Third column is the second percentage. Percentage of
    ligands that pass the second stage.
    The four last columns represent the protocol.
    All the protocols tried are written at the end.
    The ones for which time is less than 12%, perc1 is
    less than 30% and perc2 is less than 5% but bigger than 1%
    will have a series of *** after, to indicate they are good choices
    WARNING! This is a simulation based in a small set.
    The numbers are an indication, not factual values.
```

An example file would look like as follows:

```
# 3 steps as the running filters (set by the "3" in next line)
3
if - -10 SCORE.INTER 1.0 if - SCORE.NRUNS 9 0.0 -1.0,
if - -20 SCORE.INTER 1.0 if - SCORE.NRUNS 14 0.0 -1.0,
if - SCORE.NRUNS 49 0.0 -1.0,
# 1 writing filter (defined by the "1" in next line)
1
- SCORE.INTER -10,
```

In other (more understandable) words.

First, RxDock runs 3 consecutive steps:

1. Run 10 runs and check if the `SCORE.INTER` is lower than -10, if it is the case:
2. Then run 5 more runs (until 15 runs) to see if the `SCORE.INTER` reaches -20. If it is the case:
3. Run up to 50 runs to freely sample the different conformations the molecule displays.

And, second:

For the printing information, only print out all those poses where `SCORE.INTER` is better than -10 (for avoiding excessive printing).

rblist

`rblist` – Output interaction center info for ligands in SD file (with optional autoionisation).

```
rblist -i <InputSDFile> [-o <OutputSDFile>] [-ap ] [-an] [-allH]

-i <InputSDFile> - input ligand SD file
-o <OutputSDFilde> - output SD file with descriptors (default = no output)
-ap - protonate all neutral amines, guanidines, imidazoles (default = disabled)
-an - deprotonate all carboxylic, sulphur and phosphorous acid groups (default =␣
↪disabled)
-allH - read all hydrogens present (default = polar hydrogens only)
-tr - rotate all secondary amides to trans (default = leave alone)
-l - verbose listing of ligand atoms and rotable bonds (default = compact table␣
↪format)
```

3.12 Appendix

Table 3.18: Van der Waals parameters in Tripos 5.2 force field (R = radius (Å); K = well depth (kcal/mol); IP = Ionization potential (eV); POL = polarisability (10^{25} cm^3)).

Atom Type	R	K	IP	POL	Description
H	1.5	0.042	13.6	4	Non-polar hydrogen
H.P	1.2	0.042	13.6	4	Polar hydrogen
C.3	1.7	0.107	14.61	13.8	C sp3 (0 implicit H)
C.3.H1	1.8	0.107	14.61	16.38	C sp3 (1 implicit H)
C.3.H2	1.9	0.107	14.61	19.27	C sp3 (2 implicit H)
C.3.H3	2	0.107	14.61	22.47	C sp3 (3 implicit H)
C.2	1.7	0.107	15.62	13.8	C sp2 (0 implicit H)
C.cat	1.7	0.107	15.62	13.8	C sp2 (guanidinium centre)
C.2.H1	1.8	0.107	15.62	16.38	C sp2 (1 implicit H)
C.2.H2	1.9	0.107	15.62	19.27	C sp2 (2 implicit H)
C.ar	1.7	0.107	15.62	13.8	C aromatic (0 implicit H)
C.ar.H1	1.8	0.107	15.62	16.38	C aromatic (1 implicit H)
C.1	1.7	0.107	17.47	13.8	C sp (0 implicit H)
C.1.H1	1.8	0.107	17.47	16.38	C sp (1 implicit H)
N.4	1.55	0.095	33.29	8.4	N sp3+ (cationic)
N.3	1.55	0.095	18.93	8.4	N sp3
N.pl3	1.55	0.095	19.72	8.4	N trigonal planar (non-amide)
N.am	1.55	0.095	19.72	8.4	N trigonal planar (amide)
N.2	1.55	0.095	22.1	8.4	N sp2
N.ar	1.55	0.095	22.1	8.4	N aromatic
N.1	1.55	0.095	23.91	8.4	N sp
O.3	1.52	0.116	24.39	5.4	O sp3
O.2	1.52	0.116	26.65	5.4	O sp2
O.co2	1.52	0.116	35.12	5.4	O carboxylate
S.3	1.8	0.314	15.5	29.4	S sp3
S.o	1.7	0.314	15.5	29.4	sulfoxide
S.o2	1.7	0.314	15.5	29.4	sulfone
S.2	1.8	0.314	17.78	29.4	S sp2

Continued on next page

Table 3.18 – continued from previous page

Atom Type	R	K	IP	POL	Description
P.3	1.8	0.314	16.78	40.6	
F	1.47	0.109	20.86	3.7	
Cl	1.75	0.314	15.03	21.8	
Br	1.85	0.434	13.1	31.2	
I	1.98	0.623	12.67	49	
Na	1.2	0.4			
K	1.2	0.4			
UNDEFINED	1.2	0.042			

Table 3.19: Geometrical parameters for empirical terms (a = Geometric variable; b = Ideal value; c = Tolerance on ideal value; d = Deviation at which score is reduced to zero).

Term	X[a]	X_0[b]	$X_{\min}$[c]	$X_{\max}$[d]	Description
S_{polar}	R_{12}	R + 0.05 Å	0.25 Å	0.6 Å	Distance between interaction centres
	α_{DON}	180°	30°	80°	Angle around donor H
	α_{ACC}	180°	60°	100°	Angle around acceptor
	$\alpha_{\text{C+}}$	180°	60°	100°	Angle between C+ACC vector and normal to plane of guanidinium group
	$\phi_{\text{ACC\_LP}}$	45°	15°	15°	From [RiboDock2004] Figure 2.
	$\theta_{\text{ACC\_LP}}$	0°	20°	60°	From [RiboDock2004] Figure 2.
	$\phi_{\text{ACC\_PLANE}}$	0°	60°	75°	From [RiboDock2004] Figure 2.
	$\theta_{\text{ACC\_PLANE}}$	0°	20°	60°	From [RiboDock2004] Figure 2.
S_{repul}	R_{12}	R + 1.1 Å	0.25 Å	0.6 Å	Distance between interaction centres
	α_{DON}	180°	30°	60°	Angle around donor H
	α_{ACC}	180°	30°	60°	Angle around acceptor
S_{arom}	R_{perp}	3.5 Å	0.25 Å	0.6 Å	From [RiboDock2004] Figure 3.
	α_{slip}	0°	20°	60°	From [RiboDock2004] Figure 3.

Table 3.20: Angular functions used to describe attractive and repulsive polar interactions (a = Interaction centre types; b = angular functions in equations (3.6)–(3.13)).

IC1[a]	ANG_{IC1}[b]	IC2[a]	ANG_{IC2}[b]
Attractive (S_{polar})			
DON	$f_1(\lvert\Delta\alpha_{\text{DON}}\rvert)$	ACC_LP	$f_1(\lvert\Delta\phi_{\text{ACC\_LP}}\rvert) \cdot f_1(\lvert\Delta\theta_{\text{ACC\_LP}}\rvert)$
DON	$f_1(\lvert\Delta\alpha_{\text{DON}}\rvert)$	ACC_PLANE	$f_1(\lvert\Delta\phi_{\text{ACC\_PLANE}}\rvert) \cdot f_1(\lvert\Delta\theta_{\text{ACC\_PLANE}}\rvert)$
DON	$f_1(\lvert\Delta\alpha_{\text{DON}}\rvert)$	ACC	$f_1(\lvert\Delta\alpha_{\text{ACC}}\rvert)$
M+	1	ACC_LP	$f_1(\lvert\Delta\phi_{\text{ACC\_LP}}\rvert) \cdot f_1(\lvert\Delta\theta_{\text{ACC\_LP}}\rvert)$
M+	1	ACC_PLANE	$f_1(\lvert\Delta\phi_{\text{ACC\_PLANE}}\rvert) \cdot f_1(\lvert\Delta\theta_{\text{ACC\_PLANE}}\rvert)$
M+	1	ACC	$f_1(\lvert\Delta\alpha_{\text{ACC}}\rvert)$
	1	ACC_LP	
C+	$f_1(\lvert\Delta\alpha_{\text{C+}}\rvert)$	ACC_PLANE	$f_1(\lvert\Delta\alpha_{\text{ACC}}\rvert)$
	1	ACC	
Repulsive (S_{repul})			
DON	$f_1(\lvert\Delta\alpha_{\text{DON}}\rvert)$	DON	$f_1(\lvert\Delta\alpha_{\text{DON}}\rvert)$
DON	$f_1(\lvert\Delta\alpha_{\text{DON}}\rvert)$	M+	1
DON	$f_1(\lvert\Delta\alpha_{\text{DON}}\rvert)$	C+	1
M+	1	C+	1
C+	1	C+	1
ACC_LP		ACC_LP	
ACC_PLANE	$f_1(\lvert\Delta\alpha_{\text{ACC}}\rvert)$	ACC_PLANE	$f_1(\lvert\Delta\alpha_{\text{ACC}}\rvert)$
ACC		ACC	

Table 3.21: Solvation parameters (a = Frequency of occurrence in training set).

Atom type	Description	N[a]	r_i	p_i	w_i
C_sp3	Apolar carbon sp3 with 0 implicit H	48	1.7	2.149	0.8438
CH_sp3	Apolar carbon sp3 with 1 implicit H	59	1.8	1.276	0.0114
CH2_sp3	Apolar carbon sp3 with 2 implicit H	487	1.9	1.045	0.0046
CH3_sp3	Apolar carbon sp3 with 3 implicit H	409	2	0.88	0.0064
C_sp2	Apolar carbon sp2 with 0 implicit H	10	1.72	1.554	0.0789
CH_sp2	Apolar carbon sp2 with 1 implicit H	45	1.8	1.073	-0.0014
CH2_sp2	Apolar carbon sp2 with 2 implicit H	26	1.8	0.961	0.0095
C_sp2p	Positive charged carbon sp2	2	1.72	1.554	-0.7919
C_ar	Apolar aromatic carbon with 0 implicit H	116	1.72	1.554	0.017
CH_ar	Apolar aromatic carbon with 1 implicit H	357	1.8	1.073	-0.0143
C_sp	Carbon sp	24	1.78	0.737	-0.0052
C_sp3_P	Polar carbon sp3 with 0 implicit H	6	1.7	2.149	-0.0473
CH_sp3_P	Polar carbon sp3 with 1 implicit H	22	1.8	1.276	-0.0394
CH2_sp3_P	Polar carbon sp3 with 2 implicit H	130	1.9	1.045	-0.0078
CH3_sp3_P	Polar carbon sp3 with 3 implicit H	69	2	0.88	0.0033
C_sp2_P	Polar carbon sp2 with 0 implicit H	57	1.72	1.554	-0.2609
CH_sp2_P	Polar carbon sp2 with 1 implicit H	30	1.8	0.961	-0.005
CH2_sp2_P	Polar carbon sp2 with 2 implicit H	1	1.8	0.961	0.0095
C_ar_P	Polar aromatic carbon with 0 implicit H	53	1.72	1.554	-0.2609
CH_ar_P	Polar aromatic carbon with 1 implicit H	34	1.8	1.073	-0.0015
H	Explicit apolar hydrogen (not used)	0	1.2	1	0
HO	Polar hydrogen bonded to O	54	1	0.944	0.0499

Continued on next page

Table 3.21 – continued from previous page

Atom type	Description	N^a	r_i	p_i	w_i
HN	Polar hydrogen bonded to N	54	1.1	1.128	-0.0242
HNp	Positively charged polar hydrogen bonded to N	23	1.2	1.049	-1.9513
HS	Polar hydrogen bonded to S	4	1.2	0.928	0.0487
O_sp3	Ether oxygen	31	1.52	1.08	-0.138
OH_sp3	Alcohol/phenol oxygen	48	1.52	1.08	-0.272
O_tri	Ester oxygen	59	1.52	1.08	0.0965
OH_tri	Acid oxygen (neutral)	6	1.52	1.08	-0.0985
O_sp2	Oxygen sp2	83	1.5	0.926	-0.1122
ON	Nitro group oxygen	18	1.5	0.926	-0.0055
Om	Negatively charged oxygen (carboxylate etc.)	7	1.7	0.922	-0.717
N_sp3	Nitrogen sp3 with 0 attached H	8	1.6	1.215	-0.6249
NH_sp3	Nitrogen sp3 with 1 attached H	11	1.6	1.215	-0.396
NH2_sp3	Nitrogen sp3 with 2 attached H	11	1.6	1.215	-0.215
N_sp3p	Nitrogen sp3+	6	1.6	1.215	-0.1186
N_tri	Amide nitrogen with 0 attached H	15	1.55	1.028	-0.23
NH_tri	Amide nitrogen with 1 attached H	8	1.55	1.028	-0.4149
NH2_tri	Amide nitrogen with 2 attached H	6	1.55	1.028	-0.1943
N_sp2	Nitrogen sp2	3	1.55	1.413	-0.0768
N_sp2p	Nitrogen sp2+	5	1.55	1.413	-0.2744
N_ar	Aromatic nitrogen	26	1.55	1.413	-0.531
N_sp	Nitrogen sp	6	1.55	1	-0.1208
S_sp3	Sulphur sp3	15	1.8	1.121	-0.0685
S_sp2	Sulphur sp2	5	1.8	1.121	-0.0314
P	Phosphorous	10	1.8	1.589	-1.275
F	Fluorine	99	1.47	0.906	0.0043
Cl	Chlorine	132	1.75	0.906	-0.0096
Br	Bromine	37	1.85	0.898	-0.0194
I	Iodine	9	1.98	0.876	-0.0189
Metal	All metals	0	0.7	1	-1.6667
UNDEFINED	Undefined types	0	1.2	1	0

CHAPTER

FOUR

USER GUIDE

4.1 Docking in 3 steps

You will find in this page a short tutorial for running RxDock.

It has been divided in 3 steps:

1. System definition
2. Cavity generation
3. Docking

4.1.1 Step 1: System definition

First of all, we need to define the system.

Below these lines you have an example for a DUD system of a typical prm file (See *Documentation* for more information):

```
RBT_PARAMETER_FILE_V1.00
TITLE gart_DUD

RECEPTOR_FILE gart_rdock.mol2
RECEPTOR_FLEX 3.0

##################################################
## CAVITY DEFINITION: REFERENCE LIGAND METHOD
##################################################
SECTION MAPPER
    SITE_MAPPER RbtLigandSiteMapper
    REF_MOL xtal-lig.sd
    RADIUS 6.0
    SMALL_SPHERE 1.0
    MIN_VOLUME 100
    MAX_CAVITIES 1
    VOL_INCR 0.0
GRIDSTEP 0.5
END_SECTION

############################
## CAVITY RESTRAINT PENALTY
############################
SECTION CAVITY
```

(continues on next page)

(continued from previous page)

```
    SCORING_FUNCTION RbtCavityGridSF
    WEIGHT 1.0
END_SECTION
```

You will need this generated `.prm` file, a receptor structure mol2 file (`gart_rdock.mol2`) and a ligand file in the cavity (`xtal-lig.sd`) for going to next stage.

Note: The receptor `.mol2` file must be preparated (protonated, charged, etc.) prior to this stage. The program chosen to do so is up to the user. As a suggestion, we usually work with MOE and/or Maestro.

4.1.2 Step 2: Cavity generation

Once the files are ready, a simple command will generate the cavity:

```
rbcavity -was -d -r <PRMFILE>
```

With the `-d` flag a grid `.grd` file is generated. This file can be visualized in a molecular viewer to check the generated cavity.

For example, in PyMOL (after loading by: `pymol <RECEPTOR>.mol2 <LIGAND>.sd <GRID>.grd`), write the following command in the console:

```
isomesh cavity, <GRID>.grd, 0.99
```

4.1.3 Step 3: Docking

Once the cavity is defined and generated, a 50 runs-per-ligand RxDock job can be run straightforwardly with the following command:

Note: The `.prm` file, receptor, reference ligand and `.as` cavity file must be in the working directory or pointed by the environmental variable `RBT_HOME`.

```
rbdock -i <INPUT>.sd -o <OUTPUT> -r <PRMFILE> -p dock.prm -n 50
```

4.2 Docking strategies

This section does not pretend to be a comprehensive user guide. It does, however, highlight the key steps the user must take for different docking strategies, and may serve as a useful checklist in writing such a guide in the future.

4.2.1 Standard docking

By standard docking, we refer to docking of a flexible, untethered ligand to a receptor in the absence of explicit structural waters or any experimental restraints.

Standard docking workflow

1. Prepare a MOL2 file for the protein or nucleic acid target, taking into account the atom typing issues described above for MOL2 file parsing. The recommendation is to prepare an all-atom MOL2 file and allow RxDock to remove the non-polar hydrogens on-the-fly.

   Important: Make sure that any non-standard atom names and substructure names are defined in `$RBT_ROOT/data/sf/RbtIonicAtoms.prm` in order for the assignment of distributed formal charges to work correctly. Make sure that the Tripos atom types are set correctly. RxDock uses the Tripos types to derive other critical atomic properties such as atomic number and hybridisation state.

   Note: The RxDock MOL2 parser was developed to read the CCDC/Astex protein `.mol2` files, therefore this validation set is the de facto standard reference. You should compare against the format of the CCDC/Astex MOL2 files if you are in doubt as to whether a particular MOL2 file is suitable for RxDock.

2. Prepare a system definition file. At a minimum, you need to define the receptor parameters, the cavity mapping parameters (`SECTION MAPPER`) and the cavity restraint penalty (`SECTION CAVITY`). Make sure you define the `RECEPTOR_FLEX` parameter if you wish to activate sampling of terminal OH and NH3+ groups in the vicinity of the docking site.
3. Generate the docking site (.as) file using `rbcavity`. You will require a reference bound ligand structure in the coordinate space of the receptor if you wish to use the reference ligand cavity mapping method.
4. Prepare the ligand SD files you wish to dock, taking into account the atom typing issues described above for SD file parsing. In particular, make sure that formal charges and formal bond order are defined coherently so that there are no formal valence errors in the file. RxDock will report any perceived valence errors but will dock the structures anyway. Note that RxDock never samples bond lengths, bond angles, ring conformations, or non-rotatable bonds during docking so initial conformations should be reasonable.
5. Run a small test calculation to check that the system is defined correctly. For example, run `rbdock` from the command line with a small ligand SD file, with the score-only protocol (`-p score.prm`) and with the `-T 2` option to generate verbose output. The output will include receptor atom properties, ligand atom properties, flexibility parameters, scoring function parameters and docking protocol parameters.
6. When satisfied, launch the full-scale calculations. A description of the various means of launching RxDock is beyond the scope of this guide.

4.2.2 Tethered scaffold docking

In tethered scaffold docking, the ligand poses are restricted and forced to overlay the substructure coordinates of a reference ligand. The procedure is largely as for standard docking, except that:

- Ligand SD files must be prepared with the `rbtether` utility to annotate each record with the matching substructure atom indices, and to transform the coordinates of each ligand so that the matching substructure coordinates are overlaid with the reference substructure coordinates. This requires a Daylight SMARTS toolkit license.

- The system definition file should contain a `SECTION LIGAND` to define which of the the ligand degrees of freedom should be tethering to their reference values. Tethering can be applied to position, orientation and dihedral degrees of freedom independently. Note that the tethers are applied directly within the chromosome representation used by the search engine (where they affect the randomisation and mutation operators), and therefore external restraint penalty functions to enforce the tethers are not required.

Important: The reference state values for each tethered degree of freedom are defined directly from the initial conformation of each ligand as read from the input SD file, and not from the reference SD file used by `rbtether`. This is why the ligand coordinates are transformed by `rbtether`, such that each ligand record can act as its own reference state. The reference SD file used by `rbtether` is not referred to by the docking calculation itself.

It follows from the above that tethered ligand docking is inappropriate for input ligand SD files that have not already been transformed to the coordinate space of the docking site, either by `rbtether` or by some other means.

Example ligand definition for tethered scaffold

This definition will tether the position and orientation of the tethered substructure, but will allow free sampling of ligand dihedrals.

```
SECTION LIGAND
   TRANS_MODE TETHERED
   ROT_MODE TETHERED
   DIHEDRAL_MODE FREE
   MAX_TRANS 1.0
   MAX_ROT 30.0
END_SECTION
```

4.2.3 Docking with pharmacophore restraints

In pharmacophore restrained docking, ligand poses are biased to fit user-defined pharmacophore points. The bias is introduced through the use of an external penalty restraint, which penalises docking poses that do not match the pharmacophore restraints. Unlike tethered scaffold docking, there is no modification to the chromosome operators themselves, hence the search can be inefficient, particularly for large numbers of restraints and/or for ligands with large numbers of matching features. Pre-screening of ligands is based purely on feature counts, and not on geometric match considerations.

The implementation supports both mandatory and optional pharmacophore restraints. The penalty function is calculated over all mandatory restraints, and over (any `NOPT` from N) of the optional restraints. For example, you may wish to ensure that any 4 from 7 optional restraints are satisfied in the generated poses.

The procedure is largely as for standard docking, except that:

- You should prepare separate pharmacophore restraint files for the mandatory and optional restraints. Note that optional restraints do not have to be defined, it is sufficient to only define at least one mandatory restraint.
- The system definition file should contain a `SECTION PHARMA` to add the pharmacophore restraint penalty to the scoring function.

4.2.4 Docking with explicit waters

Explicit structural waters can be loaded from an external PDB file, independently from the main receptor model, by adding a `SECTION SOLVENT` to the system definition file. The user has fine control over the flexibility of each water molecule. A total of 9 flexibility modes are possible, in which the translational and rotational degrees of freedom of each water can be set independently to `FIXED`, `TETHERED`, or `FREE`. Thus, for example, it is possible to define a water with a fixed oxygen coordinate (presumably at a crystallographically observed position), but freely rotating such that the orientation of the water hydrogens can be optimised by the search engine (and can be ligand-dependent).

Note: In the current implementation, solvent refers strictly to water molecules, and the format of the water PDB file is very strictly defined. In future implementations it is anticipated that other, larger (and possibly flexible) molecules will be loadable as solvent, and that other file formats will be supported.

Explicit waters workflow

1. Prepare a separate PDB file for the explicit waters according to the format prescribed (the section called *Water PDB file format*).
2. Add a `SECTION SOLVENT` to the system definition file and define the relevant flexibility parameters (Table 3.10). The minimal requirement is to define the `FILE` parameter.
3. Decide whether you wish to have different per-solvent flexibility modes (defined via the occupancy values and temperature factor values in the PDB file (Table 3.14)), or whether you wish to have a single flexibility mode applied to all waters (defined via the `TRANS_MODE` and `ROT_MODE` values in the `SECTION SOLVENT` of the receptor .prm file).

   Important: If you wish to use per-solvent flexibility modes (that is, you wish to set different modes for different waters) make sure that you do not define `TRANS_MODE` or `ROT_MODE` entries in the `SECTION SOLVENT` as these values will override the per-solvent values derived from the temperature factors in the PDB file.

   ---

4. If you have defined any waters with `TETHERED` translational or rotational degrees of freedom, define `MAX_TRANS` and/or `MAX_ROT` values as appropriate (or accept the default values). The tethered ranges are applied to all tethered waters and can not be defined on a per-solvent basis at present.

4.3 Multi-step protocol for HTVS

For high-throughput virtual screening (HTVS) applications, where computing performance is important, the recommended RxDock protocol is to limit the search space (i.e. rigid receptor), apply the grid-based scoring function and/or to use a multi-step protocol to stop sampling of poor scorers as soon as possible.

Using a multi-step protocol for the DUD system COMT, the computational time can be reduced by 7.5-fold without affecting performance by:

1. Running 5 docking runs for all ligands;
2. ligands achieving a score of -22 or lower run 10 further runs;
3. for those ligands achieving a score of -25 or lower, continue up to 50 runs.

The optimal protocol is specific for each particular system and parameter-set, but can be identified with a purpose-built script (see the *Reference guide*, section `rbhtfinder`).

Here you will find a tutorial to show you how to create and run a multi-step protocol for a HTVS campaign.

4.3.1 Step 1: Create the multi-step protocol

These are the instructions for running rbhtfinder:

```
1st) exhaustive docking of a small representative part of the
     whole library.
2nd) Store the result of sdreport -t over that exhaustive dock.
     in file  that will be the input of this
     script.
3rd) rbhtfinder <sdreport_file> <output_file> <thr1max> <thr1min> <ns1> <ns2>
     <ns1> and <ns2> are the number of steps in stage 1 and in
     stage 2. If not present, the default values are 5 and 15
     <thrmax> and <thrmin> setup the range of thresholds that will
     be simulated in stage 1. The threshold of stage 2 depends
     on the value of the threshold of stage 1.
     An input of -22 -24 will try protocols:
             5   -22     15      -27
             5   -22     15      -28
             5   -22     15      -29
             5   -23     15      -28
             5   -23     15      -29
             5   -23     15      -30
             5   -24     15      -29
             5   -24     15      -30
             5   -24     15      -31
     Output of the program is a 7 column values. First column
     represents the time. This is a percentage of the time it
     would take to do the docking in exhaustive mode, i.e.
     docking each ligand 100 times. Anything
     above 12 is too long.
     Second column is the first percentage. Percentage of
     ligands that pass the first stage.
     Third column is the second percentage. Percentage of
     ligands that pass the second stage.
     The four last columns represent the protocol.
     All the protocols tried are written at the end.
     The ones for which time is less than 12%, perc1 is
     less than 30% and perc2 is less than 5% but bigger than 1%
     will have a series of *** after, to indicate they are good choices
     WARNING! This is a simulation based in a small set.
     The numbers are an indication, not factual values.
```

Step 1, substep 1: Exhaustive docking

Hence, as stated, the first step is to run an **exhaustive docking** of a representative part of the whole desired library to dock.

For RxDock, exhaustive docking means doing **100 runs** for each ligand, whereas standard docking means 50 runs for each ligand:

```
$ rbdock -i INPUT.sd -o OUTPUT -r PRMFILE.prm -p dock.prm -n 100
```

Step 1, substep 2: `sdreport` summary

Once the exhaustive docking has finished, the results have to be saved in a **single file** and the output of the script `sdreport -t` will be used as **input for** `rbhtfinder`:

```
$ sdreport -t OUTPUT.sd > sdreport_results.txt
```

Step 1, substep 3: `rbhtfinder` script

The **last step** is to run the `rbhtfinder` script (download `sdreport_results.txt` for testing):

```
$ rbhtfinder sdreport_results.txt htvs_protocol.txt -10 -20 7 25
```

Which will result in a file called `htvs_protocol.txt`.

The parameters are explained in the script instructions. They are not always the same and as they depend on the system, you will probably have to play a little with different values in order to **obtain good parameters sets** (marked with `***` in the output).

This will happen when **time** is less than 12%, **perc1** (number of ligands that pass the first filter) is less than 30% and **perc2** (number of ligands that pass the second filter) is less than 5% but bigger than 1%.

4.3.2 Step 2: Run docking with the multi-step protocol

The script finished with two good parameters sets:

```
TIME PERC1 PERC2 N1 THR1 N2 THR2
[...]
11.928, 27.461, 3.207, 7, -12, 25, -17 ***
[...]
10.508, 18.773, 1.511, 7, -13, 25, -18 ***
[...]
```

These parameters have to be adapted to a **file** with the HTVS **protocol format** that RxDock understands.

A **template file** looks as follows (`THR1`, `THR2`, `N1` and `N2` are the parameters found above):

```
3
if - <THR1> SCORE.INTER 1.0 if - SCORE.NRUNS <N1-1> 0.0 -1.0,
if - <THR2> SCORE.INTER 1.0 if - SCORE.NRUNS <N2-1> 0.0 -1.0,
if - SCORE.NRUNS 49 0.0 -1.0,
1
- SCORE.INTER -10,
```

It is divided in 2 sections, **Running Filters** and **Writing Filters** (defined by the lines with one number).

The first line (the number 3) indicates the number of lines in the Running Filters:

- The first filter is defined as follows: if the number of runs reaches `N1` and the score is lower than `THR1`, continue to filter 2, else stop with that ligand and go to the next one.
- The second filter is defined similar to the first one: if the number of runs reaches `N2` and the score is lower than `THR2`, continue to filter 3, else stop and go to the next ligand.
- If a ligand has passed the first two filters, continue up to 50 runs.

The fifth line (the number 1 after the three Running Filters) indicates the number of lines in the Writing Filters:

- Only print out all those poses where `SCORE.INTER` is lower than -10 (for avoiding excessive printing).

For the parameters obtained in the first Section of this tutorial (first line with ⋆⋆⋆), we will have to generate a file as follows:

```
3
if - -12 SCORE.INTER 1.0 if - SCORE.NRUNS 6 0.0 -1.0,
if - -17 SCORE.INTER 1.0 if - SCORE.NRUNS 24 0.0 -1.0,
if - SCORE.NRUNS 49 0.0 -1.0,
1
- SCORE.INTER -10,
```

Please note that the parameters `N1` and `N2` are 7 and 25 but we write 6 and 24, respectively, as stated in the template.

Finally, **run** RxDock changing the flag `-n XX` for `-t PROTOCOLFILE.txt`:

```
$ rbdock -i INPUT.sd -o OUTPUT -r PRMFILE.prm -p dock.prm -t PROTOCOLFILE.txt
```

4.4 Calculating ROC curves

(Original entry published in CBDD Research Group Blog.)

Here you will find a a short tutorial about how to generate receiver operating characteristic (ROC) curves and other statistics after running RxDock molecular docking (for other programs such as Vina or Glide, just a little modification on the way `dataforR_uq.txt` file is interpreted will make it work, see below).

I assume all of you are familiar with what ROC curves are, what are they for and how they are made.

Just in case, a very brief summary would be:

- ROC curves are graphic representations of the relation existing between the sensibility and the specificity of a test. It is generated by plotting the fraction of true positives out of the total actual positives versus the fraction of false positives out of the total actual negatives.
- In our case, we will use it for checking whether a docking program is able to select active ligands with respect to inactive ligands (decoys) and whether it is able to select these active ligands in the top % of a ranked database.
- R Library ROCR is mandatory (try with command `install.packages("ROCR")` in R before downloading from source).

The example selected for this tutorial is a system from the DUD benchmark set, “hivpr” or “hiv protease”.

These are the files you will need (all can be downloaded in this Dropbox shared folder):

- List of active ligands (`ligands.txt`)
- List of inactive ligands (`decoys.txt`)
- Output file with the docked poses of each ligand with the corresponding docking scores (`hivpr_all_results.sd.gz`)
- R script with all the R commands in this tutorial (`ROC_curves.R`)

Before getting into R, the resulted docked poses have to be filtered out for only having the best pose for each ligand (the smallest score – or highest in negative value). To do so run:

```
gunzip hivpr_all_results.sd.gz
sdsort -n -s -fSCORE hivpr_all_results.sd | sdfilter -f'$_COUNT == 1' > hivpr_
↪1poseperlig.sd
# sdsort with -n and -s flags will sort internally each ligand by increasing
```

(continues on next page)

(continued from previous page)

```
# score and sdfilter will get only the first entry of each ligand.

sdreport -t hivpr_1poseperlig.sd | awk '{print $2,$3,$4,$5,$6,$7}' > dataforR_uq.txt
# sdreport will print all the scores of the output in a tabular format and,
# with command awk, we will format the results.
```

Note: `sdsort` and `sdreport` are really useful tools for managing sd formatted compound collections. They are very user-friendly and free to download. They are provided along with rDock software in the *Download* section of the website.

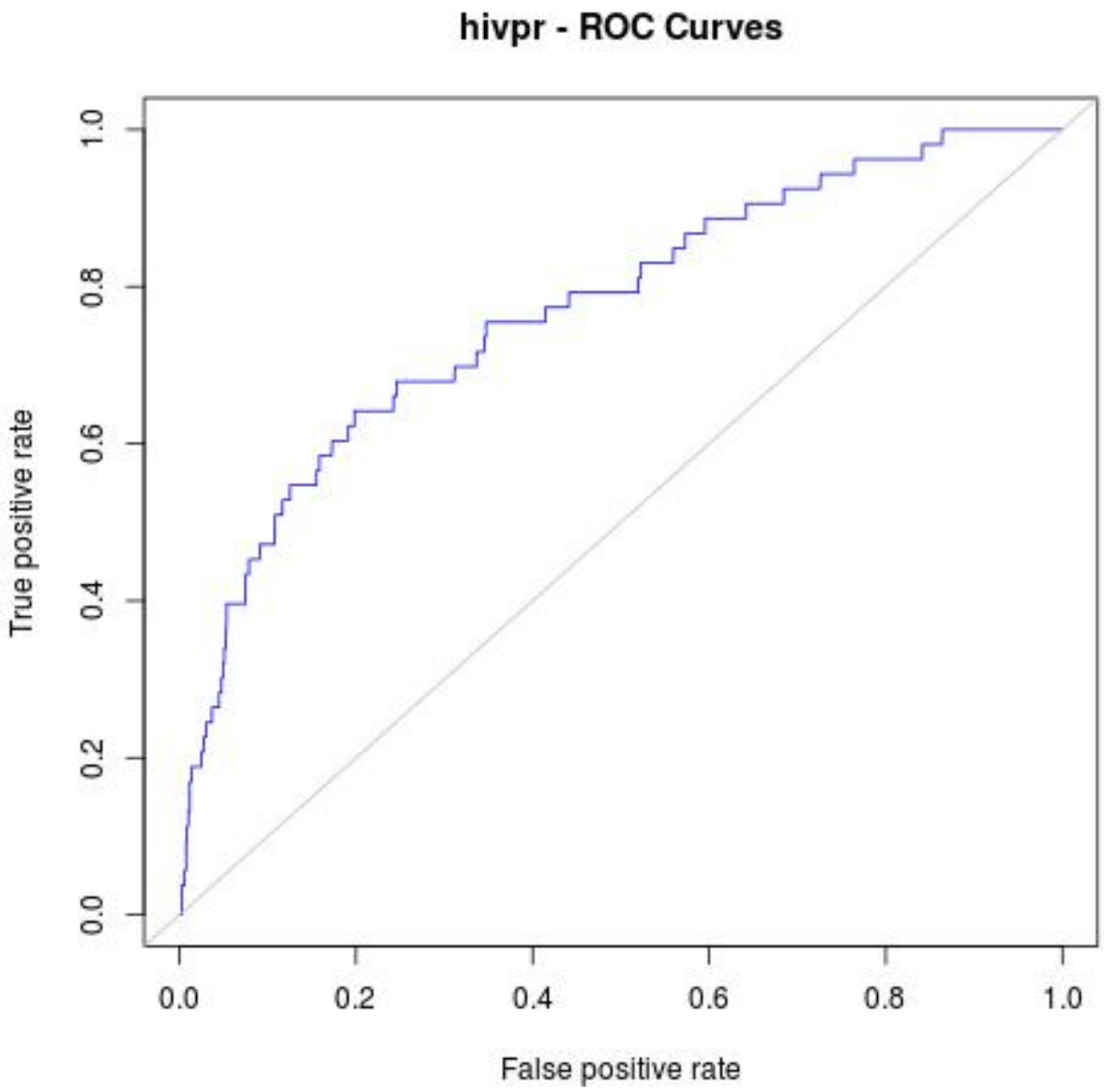
This `dataforR_uq.txt` (also in the Dropbox folder) file must contain one entry per ligand with the docked scores (what R will use to rank and plot the ROC curves).

4.4.1 R commands for generating ROC curves

Then, run the following commands in R for plotting the ROC curves:

```
# load ROCR
library(ROCR);

# load ligands and decoys
lig <- unique(read.table("ligands.txt")[,1]);
dec <- unique(read.table("decoys.txt")[,1]);

# load data file from docking
uniqRes <- read.table("dataforR_uq.txt", header=T);

# change colnames
colnames(uniqRes)[1]="LigandName";

# add column with ligand/decoy info
uniqRes$IsActive <- as.numeric(uniqRes$LigandName %in% lig)

# define ROC parameters
# here INTER is selected to compare between ligands using SCORE.INTER
# this could be changed for also running with other programs
predINTERuq <- prediction(uniqRes$INTER*-1, uniqRes$IsActive)
perfINTERuq <- performance(predINTERuq, 'tpr', 'fpr')

# plot in jpg format with a grey line with theoretical random results
jpeg("hivpr_Rinter_ROC.jpg")
plot(perfINTERuq, main="hivpr - ROC Curves", col="blue")
abline(0, 1, col="grey")
dev.off()
```

Which will give us the following plot:



Afterwards, other useful statistics such as AUC or Enrichment factors can also be calculated:

```
# AUC (area under the curve)
auc_rdock <- performance(predINTERuq, "auc")
auc.area_rdock <- slot(auc_rdock, "y.values")[[1]]
cat("AUC: \n")
cat(auc.area_rdock)
cat("\n\n")
```

```
AUC:
0.7700965
```

```
# Enrichment Factors
EF_rdock <- perfINTERuq@y.values[[1]] / perfINTERuq@x.values[[1]]
```

(continues on next page)

(continued from previous page)

```
EF_rdock_1 <- EF_rdock[which(perfINTERuq@x.values[[1]] > 0.01)[1]]
EF_rdock_20 <- EF_rdock[which(perfINTERuq@x.values[[1]] > 0.2)[1]]
cat("Enrichment Factor top 1%:\n")
cat(EF_rdock_1)
cat("\n\n")
```

```
Enrichment Factor top 1%:
11.11817
```

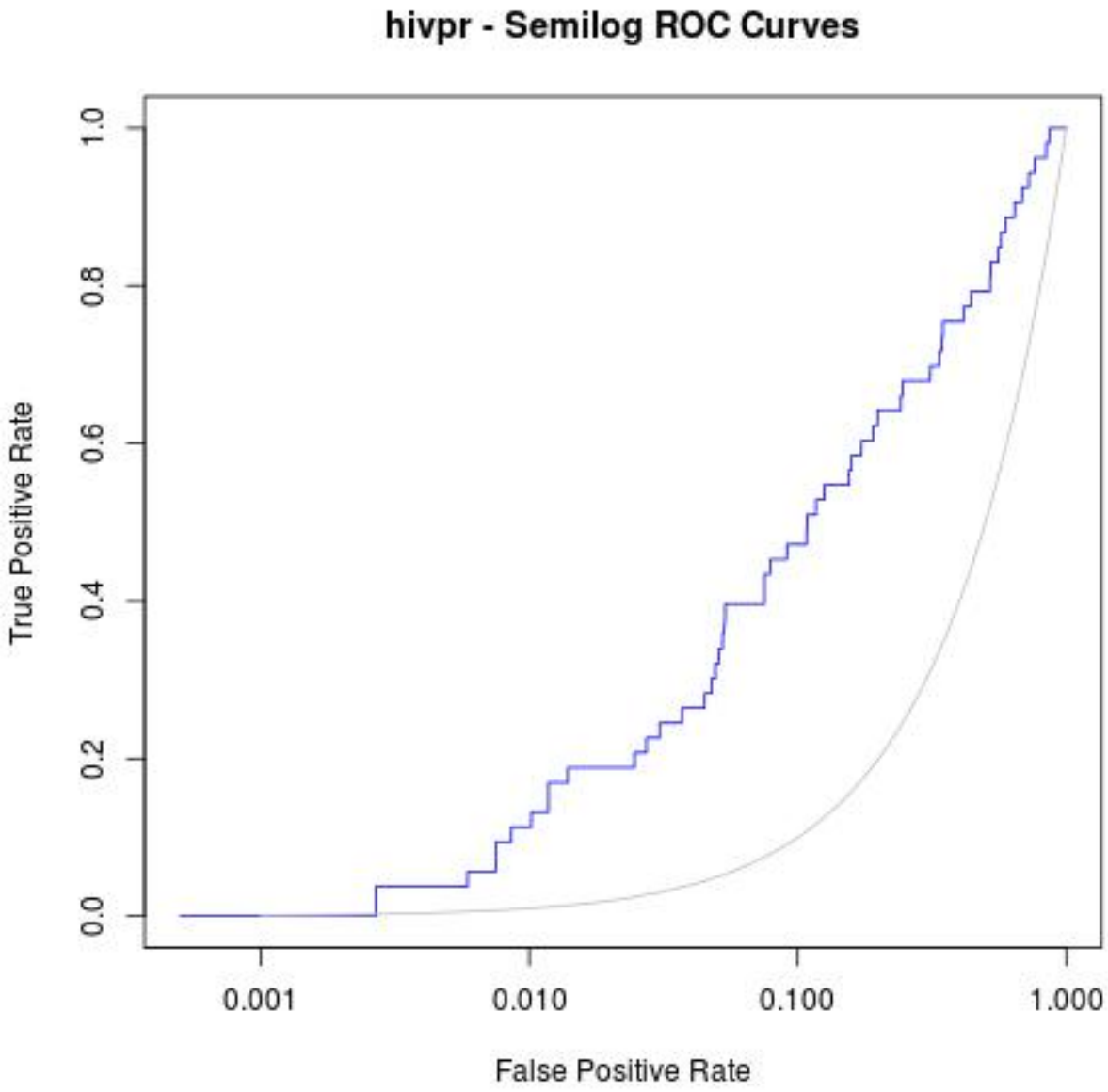
```
cat("Enrichment Factor top 20%:\n")
cat(EF_rdock_20)
cat("\n\n")
```

```
Enrichment Factor top 20%:
3.200686
```

Moreover, a good analysis of these curves is to re-plot them in semilogarithmic scale (x axis in logarithmic scale). This way, one can focus on the early enrichment of the database and have a more detailed view of the selected actives in the top % of all the ligands.

```
jpeg("hivpr_semilog_ROC.jpg")
rdockforsemilog=perfINTERuq@x.values[[1]]
rdockforsemilog[rdockforsemilog < 0.0005]=0.0005
plot(rdockforsemilog, perfINTERuq@y.values[[1]],type="l", xlab="False Positive Rate",␣
↪ylab="True Positive Rate", xaxt="n", log="x", col="blue", main="hivpr - Semilog ROC␣
↪Curves")
axis(1, c(0, 0.001, 0.01, 0.1, 1))
x<-seq(0, 1, 0.001)
points(x, x, col="gray", type="l")
dev.off()
```

Obtaining the following semi-logarithmic ROC curves:



4.5 Running docking jobs in parallel

In this short tutorial we will try to explain how to run RxDock on a computer with multiple CPUs or a cluster with different calculation nodes.

Note: RxDock has not an MPI version to be run in parallel on a computation cluster. The approach RxDock uses to parallelize the jobs is rather simple: as each molecule can be run in an independent way, the input structure file is splitted in multiple files and each of them is run independently.

For this example, we have a set of **200 molecules** (`input.sdf`) and we want to run it in **10 CPUs**.

4.5.1 Step 1: Split molecules input file

To split an SDF file (RxDock needs the input in SDF format), there is a script in RxDock package called `sdsplit` that does this.

```
$ sdsplit
Splits SD records into multiple files of equal size

Usage:  sdsplit [-<RecSize>] [-o<OutputRoot>] [sdFiles]

        -<RecSize>      record size to split into (default = 1000 records)
        -o<OutputRoot>  Root name for output files (default = tmp)

        If SD file list not given, reads from standard input
```

In our case, to split 200 molecules in 10 files (with 20 molecules each), we will have to run the following command that will generate 10 files called `split[1-10].sd`:

```
sdsplit -20 -osplit input.sdf
```

Moreover, you can use the following code which allows you to specify the number of files you want instead of the number of molecules in each file (e.g., save it in a file named `splitMols.sh`):

```
#!/bin/bash
#Usage: splitMols.sh <input> #Nfiles <outputRoot>
fname=$1
nfiles=$2
output=$3
molnum=$(grep -c '$$$$' $fname)
echo "$molnum molecules found"
echo "Dividing '$fname' into $nfiles files"
rawstep=`echo $molnum/$nfiles | bc -l`
let step=$molnum/$nfiles
if [ ! `echo $rawstep%1 | bc` == 0 ]; then
        let step=$step+1;
fi;
sdsplit -$step -o$output $1
```

To get the same as in the first case, run:

```
splitMols.sh input.sdf 10 split
```

4.5.2 Step 2: Run docking jobs with splitted files

We have two options:

- Run docking jobs locally: send it over 10 CPUs.
- Run docking jobs using a job scheduler.

Option 1: Run docking jobs locally

To run RxDock (standard mode, 50 runs per ligand) in 10 CPUs, be sure that all the necessary files are located in the working directory: receptor mol2 file, prm file, cavity as file, and reference ligand for cavity definition (if used) and run the following command:

```
for file in split*sd; do rbdock -i $file -o ${file%%.*}_out -r <PRMFILE> -p dock.prm -
↪n 50 &; done
```

This will send 10 independent docking jobs and will eventually generate 10 output files `split[1-10]_out.sd`.

So that's it, you are done!

Option 2: Run docking jobs with job scheduler

Same as in Option 1, but instead of running the command above, you have to create a queueing submission file for each of the files and submit them to the queue.

There are several options to use as a job scheduler. In our particular case, we use SGE and a typical submission file looks like this:

```
#!/bin/sh
#$ -N rdock_job1
#$ -S /bin/sh
#$ -q serial
#$ -o out.log
#$ -e err.log
#-cwd
export RBT_ROOT=/data/soft/rdock/2006.1
export LD_LIBRARY_PATH=$RBT_ROOT/lib
#next is optional
export RBT_HOME=/path/to/job/files

# These are the comands to be executed.
cd /path/to/job/files
$RBT_ROOT/bin/rbdock -i <INPUT>.sd -o <OUTPUT> -r <PRMFILE> -p dock.prm -n 50
```

This is highly recommended for running docking jobs of big molecule libraries.

For example, to run a Virtual Screening Campaign of a million compounds, you can split the molecules in 10000 files in order to have individual files with 100 molecules each and use a job scheduler to control their execution.

4.6 Pharmacophoric restraints

In this short tutorial you will find how to prepare and run Docking with pharmacophoric restraints.

Note: RxDock assumes the user knows how to compute and find pharmacophores. The user will need the coordinates, tolerance and type of restraint, which will be the input for RxDock.

4.6.1 Step 1: Pharmacophoric restraints file

The first step is to create the input file for RxDock with the necessary info.

As you can find in the *Reference guide*, this file needs one line per pharmacophore with the next structure (each element separated with a space):

```
x y z (coords of restraint centre), tolerance radius (in Angstroms), restraint type␣
↪(string)
```

The pharmacophore types accepted by RxDock are the following:

String	Description	Matches
Any	Any atom	Any non-hydrogen atom
Don	H-Bond donor	Any neutral donor hydrogen
Acc	H-Bond acceptor	Any neutral acceptor
Aro	Aromatic	Any aromatic ring centre (pseudo atom)
Hyd	Hydrophobic	Any C or S sp3, any C od S not bonded to O sp2, any Cl, Br, I
Hal	Hydrophobic, aliphatic	Subset of Hyd, sp3 atoms only
Har	Hydrophobic, aromatic	Subset of Hyd, aromatic atoms only
Ani	Anionic	Any atom with negative distributed formal charge
Cat	Cationic	Any atom with positive distributed formal charge

A sample file (`pharma.restr`) has been created containing two restraints (`Acc` and `Hyd`) with a tolerance radius of 2 and located at points (-1.75, 1.25, 0.25) and (-2, 2, -3) respectively.

```
-1.75 1.25 0.25 2.0 Acc
-2.00 2.00 -3.0 2.0 Hyd
```

4.6.2 Step 2: RxDock system definition file

The second and final step is to modify the system definition file (`FILE.prm`) to take into account the defined restraints.

Just add a `SECTION PHARMA` (see the *Reference guide* for more info) with the following lines:

```
SECTION PHARMA
    SCORING_FUNCTION RbtPharmaSF
    WEIGHT 1.0
    CONSTRAINTS_FILE pharma.restr
END_SECTION
```

With the `FILE.prm` finally being:

```
RBT_PARAMETER_FILE_V1.00
TITLE title

RECEPTOR_FILE receptor_file.mol2
RECEPTOR_FLEX 3.0

##############################################
## CAVITY DEFINITION: REFERENCE LIGAND METHOD
##############################################
SECTION MAPPER
    SITE_MAPPER RbtLigandSiteMapper
```

(continues on next page)

(continued from previous page)

```
    REF_MOL reference.sdf
    RADIUS 5.0
    SMALL_SPHERE 1.0
    MIN_VOLUME 100
    MAX_CAVITIES 1
    VOL_INCR 0.0
    GRIDSTEP 0.5
END_SECTION

############################
## CAVITY RESTRAINT PENALTY
############################
SECTION CAVITY
    SCORING_FUNCTION RbtCavityGridSF
    WEIGHT 1.0
END_SECTION

############################
## PHARMACOPHORIC RESTRAINTS
############################
SECTION PHARMA
    SCORING_FUNCTION RbtPharmaSF
    WEIGHT 1.0
    CONSTRAINTS_FILE pharma.restr
END_SECTION
```

Note: This `FILE.prm` is an example file for the sake of the tutorial. The point here is to clarify how to define pharmacophoric restraints and how to configure RxDock to take them into account.

Finally, when running RxDock, the user can check if the program writes similar lines as the following to be sure that the restraints have been correctly read.

```
[...]
RbtPharmaSF: Reading mandatory ph4 constraints from /path/to/pharma.restr
(-1.75, 1.25, 0.25)  2.0     Acc
(-2.0, 2.0, -3.0)    2.0     Hyd
RbtPharmaSF: No optional ph4 constraints file found
[...]
```

4.6.3 Step 3: Optional constraints

This tutorial is an example for mandatory constraints. Optional constraints can also be configured in a different file (same format as pharma.restr created above). The `SECTION PHARMA` in the RxDock System Configuration File should be modified as follows (NOPT=1 means that only one of the optional restraints has to be met):

```
SECTION PHARMA
    SCORING_FUNCTION RbtPharmaSF
    WEIGHT 1.0
    CONSTRAINTS_FILE pharma.restr
    OPTIONAL_FILE optional_pharma.restr
    NOPT 1
END_SECTION
```

Tip: For more information about the pharmacophoric restraints and the parameters in `SECTION PHARMA`, please go to the *Reference guide*.

CHAPTER

FIVE

DEVELOPER GUIDE

5.1 Target platforms

5.1.1 Primary and secondary target platforms

Primary target platforms are:

- Linux on AMD64 (x86-64) with the GCC compiler
- FreeBSD on AMD64 with the Clang compiler

These platforms are used for RxDock development on an ongoing basis. Specifically, the target operating system and compiler versions are the latest long-term supported releases of the main Linux distributions (e.g. Red Hat Enterprise Linux and CentOS, SUSE Linux Enterprise and openSUSE Leap, Ubuntu) and the latest point release of FreeBSD-STABLE.

Secondary target platforms are:

- macOS on x86-64 with Xcode (Clang compiler)
- Windows on x86-64 with MinGW (GCC compiler)

The feature set on the secondary target platforms will match the primary platforms and bugs that affect these platforms will be treated as release blockers.

5.1.2 Tertiary and quaternary target platforms

Tertiary target platforms are:

- Windows on x86-64 with MSVC compiler
- Linux on POWER8 and POWER9 (both big- and little-endian: power64, ppc64, power64le, ppc64le) with GCC and Clang compilers
- other modern Unix-like operating systems (e.g. Illumos) with the GCC and Clang compilers
- other compilers on Linux, FreeBSD, macOS, and Windows (e.g. PGI)

Bugs that affect the tertiary target platforms will be fixed if feasible. The care will be taken to make sure that the code compiles and that the basic functionality works. Realistically, most of the features available on the primary and the secondary target platforms will be provided on these platforms as well, but no promises will be made in advance.

Quaternary target platforms are:

- Linux, FreeBSD, macOS, and Windows on 32-bit x86
- Linux and FreeBSD on ARMv7 (armhf, armhfp) and ARMv8 (arm64, aarch64)

- Linux and FreeBSD on POWER7 and older processors

Bugs that affect the quaternary target platforms will be considered for fixing if an interested user contributes a patch.

5.2 Build system

RxDock uses the Meson build system.

5.3 Coding standards

RxDock is written using the LLVM style C++ code and formatted using ClangFormat.

5.3.1 LLVM coding standards and additions

The coding standards are documented in LLVM Coding Standards. With regards to that document, RxDock has made the following changes in the coding standards:

- Standard C++11 is used instead of C++14. In the near future the switch will be made to C++17.
- Implementation file extension should be `.cxx` instead of `.cpp`.
- Variable names should use `camelBack` instead of `CamelCase` and member variables should be prefixed with `m_` (similar to the proposed Variable Names Plan in LLVM).

5.3.2 See also

Helpful online sources of knowledge about C++:

- News, Status and Discussion about Standard C++
- C++ reference
- C++ subreddit (/r/cpp)

Particularly useful books about software design, algorithms, and programming:

1. Code Complete by Steve McConnel, published by Microsoft Press in 2004.
2. Metaheuristics: From Design to Implementation by El-Ghazali Talbi, published by Wiley in 2009.
3. Effective Modern C++ by Scott Meyers, published by O'Reilly in 2014. Also interesting and useful are "Effective C++", "More Effective C++", and "Effective STL" by the same author.

The future development will switch from the use of C++11 to the use of C++17. A very good book covering the new features in C++17 is C++17 in Detail by Bartlomiej Filipek, published by Leanpub in 2017. The blog post C++17 Features is a shorter version.

5.4 Documentation

5.4.1 General documentation

General documentation, including man pages, is built using Sphinx.

5.4.2 Codebase documentation

The codebase is documented using Doxygen.

5.5 Versioning

5.5.1 Version control system

RxDock source code is kept in a Git repository.

5.5.2 Version numbering

RxDock uses semantic versioning.

CHAPTER

SIX

SUPPORT

If you are having some trouble regarding usage, compilation, development or anything else, you can use different options to ask for support.

6.1 Mailing lists

If you are having any kind of trouble, you have any questions or anything related to general usage of the program please search and use our the discussion section of RxDock channel on Telegram.

6.2 Issue tracker

Mostly for developers and code-related problems. If you find a bug, please report it in issues section in RxDock GitLab project.

BIBLIOGRAPHY

[rDock2014] Ruiz-Carmona, S., Alvarez-Garcia, D., Foloppe, N., Garmendia-Doval, A. B., Juhos S., et al. (2014) rDock: A Fast, Versatile and Open Source Program for Docking Ligands to Proteins and Nucleic Acids. PLoS Comput Biol 10(4): e1003571. doi:10.1371/journal.pcbi.1003571

[RiboDock2004] Morley, S. D. and Afshar, M. (2004) Validation of an empirical RNA-ligand scoring function for fast flexible docking using RiboDock®. J Comput Aided Mol Des, 18: 189–208. doi:10.1023/B:JCAM.0000035199.48747.1e

[ASTEX2007] Hartshorn, M.J., Verdonk, M.L., Chessari, G., Brewerton, S.C., Mooij, W.T.M., Mortenson, P.N., and Murray, C.W. (2007). Diverse, High-Quality Test Set for the Validation of Protein-Ligand Docking Performance. J. Med. Chem. 50, 726–741. doi:10.1021/jm061277y

[GOLD2005] Verdonk, M.L., Chessari, G., Cole, J.C., Hartshorn, M.J., Murray, C.W., Nissink, J.W.M., Taylor, R.D., and Taylor, R. (2005). Modeling Water Molecules in Protein-Ligand Docking Using GOLD. J. Med. Chem. 48, 6504–6515. doi:10.1021/jm050543p

[PDBbind2004] Wang, R., Fang, X., Lu, Y., and Wang, S. (2004). The PDBbind Database: Collection of Binding Affinities for Protein-Ligand Complexes with Known Three-Dimensional Structures. J. Med. Chem. 47, 2977–2980. doi:10.1021/jm030580l

[WSAS2001] Wang, J., Wang, W., Huo, S., Lee, M., and Kollman, P.A. (2001). Solvation Model Based on Weighted Solvent Accessible Surface Area. J. Phys. Chem. B 105, 5055–5067. doi:10.1021/jp0102318

[CORINA1990] Gasteiger, J., Rudolph, C., and Sadowski, J. (1990). Automatic generation of 3D-atomic coordinates for organic molecules. Tetrahedron Computer Methodology 3, 537–547. 10.1016/0898-5529(90)90156-3

[RASASA1988] Hasel, W., Hendrickson, T.F., and Still, W.C. (1988). A rapid approximation to the solvent accessible surface areas of atoms. Tetrahedron Computer Methodology 1, 103–116. doi:10.1016/0898-5529(88)90015-2