Phase 3.4 Command Line Tutorial: Creating and Searching 3D Databases

This tutorial demonstrates how 3D databases may be created, modified and searched by running command line tools that accompany Phase version 3.4, which is part of the Schrödinger Suite 2012 release.

Phase databases created in this tutorial are stored in the same format as those created using the Maestro graphical interface. Users may therefore employ the graphical interface to modify and search a Phase database created using the tools described in this tutorial.

Phase has officially transitioned to a new, more compact database storage format, and the following tools are now deprecated:

```
$SCHRODINGER/phase_dbsearch
$SCHRODINGER/phase_fileSearch
$SCHRODINGER/phase_gridSearch
$SCHRODINGER/utilities/pharm_align_mol
$SCHRODINGER/utilities/phasedb_check
$SCHRODINGER/utilities/phasedb_confsites
$SCHRODINGER/utilities/phasedb_convert
$SCHRODINGER/utilities/phasedb_count_records
$SCHRODINGER/utilities/phasedb_export
$SCHRODINGER/utilities/phasedb_fetch_hits
$SCHRODINGER/utilities/phasedb_findmatches
$SCHRODINGER/utilities/phasedb_index
$SCHRODINGER/utilities/phasedb_manage
$SCHRODINGER/utilities/phasedb_match_keys
$SCHRODINGER/utilities/phasedb_props
$SCHRODINGER/utilities/phasedb_split_records
$SCHRODINGER/utilities/phasedb_subset
```

These tools will be eliminated from the Schrödinger Suite in 2013.

1.1 Conventions

Shell commands that the user should type will be preceded by the "`Shell>`" prompt and they will appear in fixed font, e.g.,

```
Shell> cd db_tutorial
```

Parameters for which the user must supply a value will appear in italics and in brackets, e.g.,

```
-i <inFile>
```

Important information about changes from previous versions of the software will be indented and italicized:

> *Users who are familiar with the Phase 3.3 command line database tools may recognize that a new option…*

1.2 Getting Started

In order to run the programs described in this tutorial, Phase 3.4 must be installed on the machine you are currently using, as well as on any remote hosts on which jobs will be run.[1] You must also define the `SCHRODINGER` environmental variable so that it points to the directory in which the Schrödinger software is installed.

A Phase 3D database must reside in a filesystem that is uniformly accessible to all machines on the network that will need to access the database, whether for purposes of creation, modification, or searching. Thus if a job is launched from *host1* and run on 10 processors that reside on *host2*, all 10 CPUs on *host2* must have access to the database via the same absolute path. Note that *host1* is not required to have access to the database, although the present tutorial assumes that the database filesystem is mounted locally.

To create a database, you must have a collection of molecules stored in one or more Maestro or SD files, which may be compressed via gzip. Valid file extensions are .mae, .mae.gz, .maegz, .sdf, .sd, .sdf.gz, and .sd.gz. Each molecule may be represented by a single low-energy 3D conformer or by multiple 3D conformers. In the former case, you may import the single-conformer molecules into the database and create multi-conformer models later. If you are starting with 2D structures or SMILES strings, you must create low-energy 3D models first, preferably using LigPrep.

Each Maestro or SD file may contain multiple molecules, but if multiple conformers are present, they must be grouped in consecutive CT blocks. In the case of Maestro files, a multi-conformer molecule may be stored as a full CT block (`f_m_ct`) followed by partial CT blocks (`p_m_ct`), or it may be stored as a set of consecutive full CT blocks.

1.3 Database Programs

The following *top-level* programs should be present in the directory `$SCHRODINGER`:

`phase_database`	– Performs all tasks related to Phase database creation and management.
`phase_find_matches`	– Searches a Phase database, one or more structure files, or a Phase command line project for matches to a pharmacophore hypothesis.

[1] Schrödinger software is frequently installed on a single filesystem that is accessible to multiple hosts, so strictly speaking, not every host must have its own separate installation.

These two programs comprise the entire command line interface for Phase database creation and searching.

1.4 Creating a New 3D Database

Place a copy of the file `db_tutorial.tar.gz` in a directory located in a uniformly accessible filesystem (as discussed in Section 1.2), and extract the file as follows:

```
Shell> gunzip -c db_tutorial.tar.gz | tar xf -
```

Move to the `db_tutorial/` directory, which should be created automatically by the extraction:

```
Shell> cd db_tutorial
```

For future convenience, we shall define an environment variable `TPATH` to hold the absolute path to the tutorial directory:

csh/tcsh:
```
Shell> setenv TPATH `pwd`
```

sh/bash/ksh:
```
Shell> export TPATH=`pwd`
```

Note that single back-quotes are used to surround the `pwd` command. Verify that `TPATH` has been assigned correctly:

```
Shell> echo $TPATH
```

This should print the absolute path to the current directory. Note that if you require more than one terminal session to complete the tutorial, or if you switch shells or windows, you will need to redefine `TPATH`.

Before running any command line program, it's a good idea to simply type the name of that program to see the usage message. You will be creating the database using `phase_database`, so type the following command:

```
Shell> $SCHRODINGER/phase_database
```

Observe that a *<task>* argument must be supplied to indicate what sort of operation is to be performed. To see a shortened help message with options for only a particular task, invoke `phase_database` with the flag `-help_<task>` (e.g., `-help_import`, `-help_revise`, etc.). To see a series of examples showing how to perform various tasks, use `-help_examples`. Please examine the help messages in detail so that you are familiar with the wide variety of tasks and options supported by `phase_database`.

Now create a new database named `stdDB.phdb` in the directory `$TPATH`, and populate it with 100 single-conformer molecules contained in `$TPATH/userFiles/singleConfs.mae.gz`:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb import -new \
       -i $TPATH/userFiles/singleConfs.mae.gz -verbose
```

One obvious change from the previous database format is that it is no longer necessary to create a root directory to hold the database files. The root directory is stdDB.phdb, which is created automatically by phase_database when the -new flag is supplied. Furthermore, $TPATH/stdDB.phdb is the fully qualified name of the database, and that convention is used whether running through the command line or the graphical interface.

By default, the assigned job name is `database_<task>`, so it will be `database_import` when an import is done. This convention can be overridden by supplying the optional third argument `<jobName>`.

The above command will run the job on the machine you are currently logged into, but you could direct it to be run on another machine by using the option `-HOST <host>`, where `<host>` is an entry name in the file `$SCHRODINGER/schrodinger.hosts`. As discussed in Section 1.2, the database directory must be accessible to `<host>` via the absolute path that's specified in the `phase_database` command. The database need not be accessible to the machine you are logged into, so if you specify `-HOST <host>`, you could launch the job from any machine on your network that can reach `<host>` through Schrödinger job control.

Note that only single-CPU jobs are supported when importing. Splitting the job across multiple CPUs would not offer any speedup because nearly all the time is devoted to reading the supplied Maestro or SD file. Dividing this work among multiple CPUs would still require that a master job read the entire input file and split it up for distribution. Observe that you can run `phase_database` locally as a foreground process, rather than as a Schrödinger job, by using the flag `-NOJOBID`.

The import will require only about one second to finish, after which you should see the following job files in the current directory:

File Name	Description
`database_import.log`	Log file with details of the import process.
`database_import.okay`	Created only if the job completes successfully.
`database_import_errors.out`	Error messages, if any errors occur. Empty otherwise.
`database_import_new_phase.inp`	Phase subset file with newly imported record numbers.

The tail end of the file `database_import.log` should look like the following:

```
.
.
.
LIGAND_NAME = block_1/mol_98  # Title = "852130"  Confs = 1
LIGAND_NAME = block_1/mol_99  # Title = "862412_2"  Confs = 1
```

```
LIGAND_NAME = block_1/mol_100  # Title = "862427"  Confs = 1
Committing 100 records
Updating top-level database files . . .

A total of 100 records added
Total number of records stored = 100

Removing restart directory

CPU time = 1.28 sec

phase_database successfully completed

Driver script for phase_database job database_import finished
Current time: Tue Jan 17 13:50:50 2012
Elapsed time = 00:00:01
phase_database results for database_import are complete
```

The reported CPU time of course depends on the hardware you are using.

Now verify that the appropriate files have been created in the database directory:

```
Shell> ls -F $TPATH/stdDB.phdb
```

You should see the following files:

File Name	Description
`database.sqlite`	SQLite database with tables that hold global information about the Phase database. See below for details.
`database_dbversion`	Version file. Holds the Phase version number, the method of creation (always "CL"), and the storage format ("SQLite"). Do not modify.
`database_feature.ini`	Default pharmacophore feature definitions. You may replace this file with customized definitions before creating pharmacophore sites (or you could have used the `-fd` *`<fdFile>`* option when `phase_database` was used to create the database).
`database_info.log`	Tracks all changes to the database.
`database_ligands/`	Subdirectory under which all database structure files are stored.
`database_master_phase.inp`	List of all database records (do not modify).
`database_summary.csv`	Human-readable copy of the data in `summary_table` (see below).

The file `database.sqlite` contains the following tables:

Table Name	Description
`dprop_name_table`	Holds the name of the property used to detect/eliminate duplicate structures. This is currently hard-coded to `s_phase_Unique_SMILES`.

dprop_values_table	Holds mol_id and s_phase_Unique_SMILES for each database record, where mol_id is the primary key.
props_table	This table will exist only if an "extract" job has been run. If so, it will hold mol_id, title, num_confs, has_sites, plus all imported and computed properties for each record that was in the database when the extract job was run.
props_map_table	Mappings of the column names in props_table to the CT-level property names in the block database files stored under database_ligands/. This table always exists, but it will be empty if properties have not been extracted.
props_stale_table	Contains a 1/0 value indicating whether or not the database contains records that are not in props_table. A value of 1 indicates that an "extract" job needs to be run in order to bring props_table up-to-date.
summary_table	Holds mol_id, title, num_confs, and has_sites for each database record. has_sites will be 1 or 0, depending on whether or not pharmacophore sites are stored for a given record.

At this point, we have extracted 100 single-conformer molecules from the file singleConfs.mae.gz, and we have stored them in a single block SQLite file:

```
$TPATH/stdDB.phdb/database_ligands/block_1/ block_struct_1.sqlite
```

Up to 5,000 molecule records will be stored in this file, with block_2/block_struct_2.sqlite being created when molecule 5,001 is added. You can reduce the number of molecules per block during an import by using the option -blimit <maxRec>. This is most relevant when you create a small database and you wish to split conformer and/or site generation over multiple processors. Because it is not safe to allow multiple processors to write to the same block SQLite file, each processor must operate on one or more separate blocks. By reducing the block size, a larger number of blocks will be created, and more processors may be utilized for conformation/site generation.

We could now search this database if we were willing to generate conformers and pharmacophore sites in memory without storing them in the database, a process we refer to as *flexible* searching. In the present case, however, we will be storing conformers and sites in the database. Flexible databases are covered in Sections 1.11 and 1.12.

1.5 Adding Molecules to an Existing Database

In this section, a set of multi-conformer molecules is added to the database created in the previous section. If you are resuming the tutorial in a terminal session or window that is different from that used in Section 1.4, you will need to redefine the TPATH environment variable.

The file $TPATH/userFiles/multiConfs.mae.gz contains a set of 36 endothelin ligands with pre-generated conformational models. In order to add these structures to the existing database

using `phase_database`, we omit the `-new` flag, and we must supply the flag `-multi` to indicate that the file being imported contains multiple conformers per molecule. We will also supply a unique *<jobName>* argument so that the job files from the previous import are not overwritten. Accordingly, add molecules to the database as follows:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb import add_multi \
       -i $TPATH/userFiles/multiConfs.mae.gz –multi -verbose
```

The job should finish in a just a few seconds. Once the job has completed, the tail end of the file `add_multi.log` should contain output similar to the following:

```
.
.
.
LIGAND_NAME = block_1/mol_134  # Title = "endo-34"  Confs = 8
LIGAND_NAME = block_1/mol_135  # Title = "endo-35"  Confs = 3
LIGAND_NAME = block_1/mol_136  # Title = "endo-36"  Confs = 3
Commiting 36 records
Updating top-level database files . . .

A total of 36 records added
Total number of records stored = 136

Removing restart directory

CPU time = 1.67 sec

phase_database successfully completed

Driver script for phase_database job add_multi finished
Current time: Tue Jan 17 13:52:54 2012
Elapsed time = 00:00:02
phase_database results for add_multi are complete
```

You may also examine the file `$TPATH/stdDB.phdb/database_info.log` to see the full history of changes made to the database.

1.6 Creating Conformers and Pharmacophore Sites

So far, we have created a database of 136 molecules, storing a single conformer for each of the first 100, and multiple conformers for the last 36. We now demonstrate how to create conformers for the first 100 molecules only, and pharmacophore sites for all 136. Conformers are generated using a rapid torsional sampling technique, followed by application of a soft non-bonded potential to eliminate high-energy structures.

We will be running a `phase_database` “revise” job, so type the following command to view the relevant help message:

```
Shell> $SCHRODINGER/phase_database -help_revise
```

The options `-confs all|auto`, `-sites`, and `-isub <subset>` provide a high level of control and flexibility when creating conformers and pharmacophore sites. You may use `-confs all` to force conformer and site creation for all molecules, or, if you wish to generate conformers only for molecules that are represented by a single conformer, use `-confs auto`. Note that sites are automatically generated whenever `-confs` is supplied. If `-confs` is omitted, conformer generation will be skipped altogether, and `-sites` may be used to generate pharmacophore sites for the existing conformers. The `-sub` option allows you to change the domain of operation from the entire database to any subset of molecules in the database (see Section 1.17 for more on using subsets).

Ensure that the `TPATH` environment variable is defined as described in Section 1.4, and then launch the `phase_database` "revise" job as follows:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb revise auto_confs \
       -confs auto
```

Observe that we are using default settings for conformer generation, so a maximum of 100 conformers will be created and stored for each molecule which isn't already represented by multiple conformers. When the job has successfully finished, the file `auto_confs.okay` will appear. The tail end of `auto_confs.log` should look like the following:

```
.
.
.
All subjobs successfully completed.  Updating top-level database files . . .
Revision of 136 out of 136 records completed
Average number of conformers per revised record = 44.53
Total number of records stored = 136

Removing restart directory

CPU time = 0.58 sec

phase_database successfully completed

Driver script for phase_database job auto_confs finished
Current time: Tue Jan 17 13:56:44 2012
Elapsed time = 00:02:45
phase_database results for auto_confs are complete
```

One subtle point worth mentioning is that if you generate conformers more than once for the same database records, you will generally not get the same sets of conformers created each time. This is because you are not starting with the same seed structure. The first time around, the seed structure for a given molecule is whatever structure you imported. That structure is discarded by default when the new conformers are stored in the database, so if you generate conformers for the same molecule again, the seed structure will be the first conformer stored. While the conformers generated with different seeds are usually similar, they are not the same.

1.7 Deleting Molecules from a Database

There may come a time when a user needs to delete molecules from a database, and we demonstrate how to do this by running a `phase_database` "delete" job and specifying a set of *database records* for deletion. To see the relevant help message, type the following:

```
Shell> phase_database -help_delete
```

Observe that in addition to deleting a specific set of records, `phase_database` can identify records that contain duplicate structures and delete them, or simply produce a subset file with the duplicate record numbers.

Recall that the file `$TPATH/stdDB.phdb/database_info.log` contains a full accounting of all changes made to the database, including the unique identifier for each record added to the database. You may therefore use the contents of this file to construct a subset of database records for deletion. Suppose you wanted to delete the last 10 molecules from the database. To do so, simply create a subset file with the appropriate database records as follows:

```
Shell> grep LIGAND_NAME $TPATH/stdDB.phdb/database_info.log \
       | tail -10 > last10_phase.inp
```

We are including "`_phase.inp`" at the end of the subset file name, because the Phase convention is to treat everything to the left of "`_phase.inp`" as the subset name. The contents of `last10_phase.inp` should be as follows:

```
LIGAND_NAME = block_1/mol_127  # Title = "endo-27"  Confs = 6
LIGAND_NAME = block_1/mol_128  # Title = "endo-28"  Confs = 6
LIGAND_NAME = block_1/mol_129  # Title = "endo-29"  Confs = 18
LIGAND_NAME = block_1/mol_130  # Title = "endo-30"  Confs = 6
LIGAND_NAME = block_1/mol_131  # Title = "endo-31"  Confs = 11
LIGAND_NAME = block_1/mol_132  # Title = "endo-32"  Confs = 14
LIGAND_NAME = block_1/mol_133  # Title = "endo-33"  Confs = 4
LIGAND_NAME = block_1/mol_134  # Title = "endo-34"  Confs = 8
LIGAND_NAME = block_1/mol_135  # Title = "endo-35"  Confs = 3
LIGAND_NAME = block_1/mol_136  # Title = "endo-36"  Confs = 3
```

Everything to the right of a pound sign "`#`" is treated as a comment and thus ignored. Now delete these records from the database as follows:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb delete -isub last10
```

This operation should require very little time, and the tail end of the file `database_delete.log` should look like:

```
.
.
.
Reading records from last10_phase.inp
```

```
Deleting 10 records . . .

Total number of records stored = 126

CPU time = 0.63 sec

phase_database successfully completed

Driver script for phase_database job database_delete finished
Current time: Tue Jan 17 14:01:48 2012
Elapsed time = 00:00:01
phase_database results for database_delete are complete
```

The database now contains only 126 molecules.

1.8 Setting Database Permissions

When you create a database, you will naturally be the owner of all the files and directories associated with that database. As such, you will normally have read/write permissions to the files, and read/write/execute permissions to the directories. You should therefore encounter no difficulties in modifying a database that you create yourself. However, you may or may not want other users to be granted this same privilege.

If you want to be certain that you are the only person who will ever modify a particular database, then you should remove write permissions for all other users throughout the database tree. By default, the files and directories you create will normally not carry write permissions for other users (unless you change the default umask), but we demonstrate how to explicitly set those permissions here. First ensure that the `TPATH` environment variable is defined as described in Section 1.4, and then issue the following command:

```
Shell> chmod –R g-w,o-w $TPATH/stdDB.phdb
```

Write permissions for other users have now been removed recursively throughout the database, and you are the only person who can modify the database.

1.9 Database Searching: Background

In this section, material is presented that is critical to understanding how Phase 3D database searching works. Users are strongly encouraged to read this section so that they are familiar with terminology used in subsequent sections.

Before setting up and running a database search, you normally want to be in the directory where the pharmacophore hypothesis files reside. Ensure that the `TPATH` environment variable is defined as described in Section 1.4, then move to the directory containing the hypothesis files:

```
Shell> cd $TPATH/userFiles
```

This directory contains the following hypothesis-related files, which were created from a set of endothelin ligands:

File Name	Description
`DHHRR_9.def`	Pharmacophore feature definitions used to develop the hypothesis.
`DHHRR_9.mae`	Contains the reference ligand conformation, i.e., the full 3D chemical structure of the molecule that gave rise to the hypothesis.
`DHHRR_9.qsar`	QSAR model.
`DHHRR_9.tab`	General information about the hypothesis.
`DHHRR_9.xvol`	Excluded volume definitions.
`DHHRR_9.xyz`	Hypothesis site point coordinates.

The QSAR model file and excluded volumes file will be present only if these characteristics have been defined for the hypothesis. All hypothesis files except `DHHRR_9.qsar` contain ordinary text, so you may examine their contents, but do not modify them.

When a Phase 3D database is searched, the process is formally divided into two steps: finding matches and scoring hits.

When finding matches, the pharmacophore sites of a database molecule are searched for geometric arrangements of points that match the hypothesis with respect to feature types, intersite distances, and positional tolerances. For example, the DHHRR_9 hypothesis contains one donor (D), two hydrophobes (H1, H2), and two aromatic rings (R1, R2). These five pharmacophore site points give rise to 10 unique intersite distances: d_{DH1}, d_{DH2}, d_{DR1}, d_{DR2}, d_{H1H2}, d_{H1R1}, d_{H1R2}, d_{H2R1}, d_{H2R2}, and d_{R1R2}. Accordingly, a database molecule would be scanned for occurrences of five sites of the type D, H, H, R, R, for which the 10 unique intersite distances are sufficiently close to those of the hypothesis. When such an occurrence is found, a least-squares alignment is performed to superimpose the matching site points in the database molecule onto the corresponding sites in the hypothesis, and positional tolerances are applied to determine whether pairs of aligned sites fall within a predefined radius of each other. If so, this *match* is stored in memory.

Once all matches have been recorded for a given molecule, the full 3D chemical structure of the conformer associated with each match is retrieved and aligned to the hypothesis according to the same transformation that was used to align the sites. We refer to these as *hits*, and there are various parameters and options that control which hits will survive, given a set of matches.

First and foremost, hits are sorted in order of decreasing fitness score, a quantity that measures how well the site points in the matching conformer align to those of the hypothesis, how well the matching vector features (acceptors, donors, aromatic rings) overlay those of the hypothesis, and how well the 3D chemical structure of the hit superimposes, in an overall sense, with the reference ligand conformation. Fitness is calculated as follows:

```
Fitness = alignWeight*(1.0 - alignScore/alignCutoff)
        + vectorWeight*vectorScore + volumeWeight*volumeScore
```

The various fitness score terms are defined as follows:

Term	Definition
`alignScore`	The RMS deviation between the positions of the site points in the matching conformation and the site points in the hypothesis.
`alignCutoff`	User-adjustable parameter. Default = 1.2.
`alignWeight`	User-adjustable parameter. Default = 1.0.
`vectorScore`	Average cosine between vector features in the matching conformation and vector features in the reference conformation.
`vectorWeight`	User-adjustable parameter. Default = 1.0.
`volumeScore`	$O(A,B)/[O(A,A) + O(B,B) - O(A,B)]$, where $O(A,B)$ is an approximate overlapping volume between matching conformation A and reference conformation B, and $O(A,A)$ and $O(B,B)$ are approximate self-overlaps. Each structure is represented as a collection of heavy atom van der Waals spheres.
`volumeWeight`	User-adjustable parameter. Default = 1.0.

If partial matching is invoked (see *<minSites>*, Sections 1.10 and 1.13) and a conformer fails to match all sites in the hypothesis, a penalty is applied to `alignScore`. If m out of n sites are matched, `alignScore` is modified as follows:

$$\texttt{alignScore} \rightarrow \text{sqrt}(w_{old}\cdot\texttt{alignScore}^2 + w_{new}\cdot\texttt{alignPenalty}^2)$$

where

$w_{old} = m/n$
$w_{new} = (n - m)/n$
`alignPenalty` = User-adjustable parameter. Default = 1.2.

By adjusting parameters in the fitness function, the user can control the order in which hits are returned. For example, to emphasize alignment of vector features, the `vectorWeight` parameter should be increased. If overall molecular superposition is most important, then `volumeWeight` should be increased. Note that the volume term provides a means of forcing the shape of the hit to resemble that of the reference conformation. Since the user can limit the number of hits from each molecule and the total number of hits, the fitness function ultimately controls which hits the user sees.

Application of excluded volumes also affects the survival of hits, and the user has the option of ignoring excluded volumes if they happen to exist for a given hypothesis. Although a QSAR model does not impact which hits are obtained, it can be applied or ignored.

In previous releases, a "find+fetch" job could be run, which resulted in the creation of a match file. If the user wished to change only hit treatment options without having to find matches all over again, a subsequent "fetch" job could be run against the match file. However, the creation of Phase databases containing many millions of compounds is increasingly common, and match files produced from such databases may be exceedingly

large. Moreover, users rarely performed of fetch-only searches, so the new software does not create a match file, nor does it support fetch-only searches.

1.10 Searching Existing Conformers and Sites

In this section we demonstrate how to run a normal database searching job against the database created in Sections 1.4-1.7, so you need to complete those sections if you have not done so already.

Databases are searched using `phase_find_matches`, so type the name of that program with no arguments to see the help message:

```
Shell> $SCHRODINGER/phase_find_matches
```

Most options have reasonable defaults, so you need not be concerned with everything you see in the help message, although it's still a good idea to familiarize yourself with the capabilities of `phase_find_matches`.

The most obvious change from previous releases is that it is no longer necessary to run a separate utility (phasedb_findmatches) to setup the database searching job. Instead, all options are supplied directly to the program that does the searching.

While residing in the directory `$TPATH/userFiles/`, issue the following command to launch a basic search against existing conformers and sites:

```
Shell> $SCHRODINGER/phase_find_matches $TPATH/stdDB.phdb DHHRR_9 basic_search \
       -NOCHECKPOINT
```

As discussed before, the database need not be accessible to the machine you are logged into, so if you specify `-HOST <host>`, you could launch the job from any machine on your network that can reach `<host>` through Schrödinger job control.

If you are searching a large database, it is strongly recommended that you use the option `-CHECKPOINT <path>` when you launch the job, so that if the job fails to complete for any reason (a disk goes offline, a cluster node goes down, a power failure occurs, etc.), you will be able to restart it by using `-RESTART <path>`. Note that `<path>` is the absolute path to a directory under which checkpoint files will be created and updated throughout the course of the search. You must have write permissions to this directory and it must be accessible to the host on which the job is run.

You may examine the file `basic_search.log` to monitor the progress of the job. When the job has completed, the file `basic_search.okay` will be created and you should see the following sort of output in `basic_search.log`:

```
Driver script for phase_find_matches job basic_search
Current time: Tue Jan 17 14:06:31 2012
```

```
Number of CPUs requested = 1
.
.
.
Combining subjob log files . . .

*** Contents of basic_search_sub_0.log ***
.
.
.
phase_find_matches: Tue Jan 17 14:06:31 2012

Source = <TPATH>/stdDB.phdb
Hypothesis = DHHRR_9
Job name = basic_search
Options:
-NOCHECKPOINT
-subjob basic_search_sub_0
-subproc
Assigning default positional tolerances of 2.00
Applying QSAR model in DHHRR_9.qsar
Applying excluded volumes in DHHRR_9.xvol

Pre-screening 126 database records using 2D/3D index . . .

Number of records matched = 54
Pre-screen CPU time = 0.03 sec

ID,#Confs,#Matches,#Searched
mol_1,27,33,1
mol_11,70,0,2
mol_13,28,0,3
.
.
.
mol_124,1,1,52
mol_125,1,1,53
mol_126,1,1,54

Writing 28 hits to basic_search_sub_0-hits.list

*******************************************************************************

Finished processing the current batch of 54 records

Cumulative totals:
Number of molecules searched = 54
Number of conformers searched = 2218
Number of matches found = 340
Number of molecules that produced matches = 28
Number of hits written = 28

*******************************************************************************
.
.
.
*** End of basic_search_sub_0.log ***
```

```
Combining sorted hits from completed subjobs

Importing hits from basic_search_sub_0-hits.list . . .
Number of hits stored so far = 28

Writing 28 sorted hits to basic_search-hits.maegz . . .

Hits from completed subjobs successfully combined

All subjobs successfully completed
CPU time = 0.02 sec

phase_find_matches successfully completed

Driver script for phase_find_matches job basic_search finished
Current time: Tue Jan 17 14:06:33 2012
Elapsed time = 00:00:02
phase_find_matches results for basic_search are complete
```

Observe that the pre-screen using the 2D/3D index reduced the number of records to search from 126 to 54, and that 28 of those 54 records produced hits. So while the pre-screen is not 100% effective at identifying the minimum set of records to search, it does eliminate the majority of records that cannot possibly match the hypothesis (in the present case 72 out of 98 such records were eliminated). The pre-screen is also quite fast because it requires examination of only a small amount of data per molecule.

> *In previous releases, positional tolerances were applied only when the file* `<hypoID>.tol` *or the file* `<hypoID>.dxyz` *existed. Neither of these files exists, but the log file for this job indicates that default positional tolerances of 2.0 were used, which happens to be equal to the default intersite distance matching tolerance* `<deltaDist>`*. Thus if the user doesn't define his own positional tolerances, default tolerances will be assigned from* `<deltaDist>`*. This change was made because a poor superposition of matching sites may be obtained even if* `<deltaDist>` *is satisfied (e.g., when a match corresponds to a mirror image of the hypothesis), and many users assume that a set of default positional tolerances are applied anyway as part of the matching process.*

You may view the 28 hits by importing the file `basic_search-hits.maegz` into Maestro. If you do so, you should see a number of properties that come from the database search:

Property	Description
`Unique SMILES`	Unique/canonical SMILES.
`phasedb index`	Database record number; same as `mol_id`.
`Conf Index`	Conformation index.
`Num Sites Matched`	Number of hypothesis sites matched.
`Matched Ligand Sites`	String that indicates which sites in the hit matched the hypothesis. The letter is the type of site that matched, and the number is the index of that site within the hit. If partial matching is invoked, a given site in the hypothesis may not be matched, in which case there would be a hyphen "-" rather than a number.

volumeScore	$O(A,B)/[O(A,A) + O(B,B) – O(A,B)]$, where $O(A,B)$ is an approximate overlapping volume between matching conformation *A* and reference conformation *B*, and $O(A,A)$ and $O(B,B)$ are approximate self-overlaps. Each structure is represented as a collection of heavy atom van der Waals spheres.
Align Score	Alignment score (see Section 1.9).
Vector Score	Vector score (see Section 1.9).
Volume Score	Volume score (see Section 1.9).
Fitness	Fitness score (see Section 1.9).
Pred Activity(1)	Predicted activity from 1-factor PLS QSAR model.
Pred Activity(2)	Predicted activity from 2-factor PLS QSAR model.
Pred Activity(3)	Predicted activity from 3-factor PLS QSAR model.

1.11 Creating a Flexible Database

If disk space is at a premium, a user may opt to create a database wherein only a single conformer is stored for each molecule. The necessary multi-conformer models and pharmacophore sites can be generated on-the-fly when the database is searched, without ever storing them to disk. In this section, we demonstrate how to create a database that will be searched in this manner, a so-called *flexible* database.

Ensure that the `TPATH` environment variable is defined described in section 1.4, and move to that directory:

```
Shell> cd $TPATH
```

Create a new database named `flexDB.phdb` in the directory `$TPATH`, and populate it with 100 single-conformer molecules contained in `db_tutorial/userFiles/singleConfs.mae.gz`:

```
Shell> $SCHRODINGER/phase_database $TPATH/flexDB.phdb import create_flex \
       -new -i $TPATH/userFiles/singleConfs.mae.gz -verbose
```

As in Section 1.4, this operation will require about one second to finish. Once the import has completed, the tail end of the file create_flex.log should look like:

```
.
.
.
LIGAND_NAME = block_1/mol_98  # Title = "852130"  Confs = 1
LIGAND_NAME = block_1/mol_99  # Title = "862412_2"  Confs = 1
LIGAND_NAME = block_1/mol_100  # Title = "862427"  Confs = 1
Committing 100 records
Updating top-level database files . . .

A total of 100 records added
Total number of records stored = 100

Removing restart directory
```

```
CPU time = 0.98 sec

phase_database successfully completed

Driver script for phase_database job create_flex finished
Current time: Tue Jan 17 14:15:15 2012
Elapsed time = 00:00:02
phase_database results for create_flex are complete
```

Now verify that the appropriate files have been created in the database directory:

```
Shell> ls -F $TPATH/flexDB.phdb
```

You should see the following files:

```
database.sqlite       database_info.log          database_summary.csv
database_dbversion    database_ligands/
database_feature.ini  database_master_phase.inp
```

See Section 1.4 for further details about each file. This 100-molecule database is now suitable for searching with on-the-fly generation of conformers and sites.

1.12 Running a Flexible Database Search

In this section we demonstrate how to setup and run a flexible database search, wherein conformers and sites are generated on-the-fly, but never written to disk. We will be using the database created in Section 1.11, so you need to complete that section if you have not already done so.

First move to the directory that holds the hypothesis files:

```
Shell> cd $TPATH/userFiles
```

Then issue the following command to run the job:

```
Shell> $SCHRODINGER/phase_find_matches $TPATH/flexDB.phdb DHHRR_9 flex_search \
       -flex -NOCHECKPOINT
```

The `-flex` flag triggers on-the-fly generation of conformers and sites. We are using default settings for conformer generation, just as we did when creating the multi-conformer database in Section 1.6.

When the job finishes, the log file `flex_search.log` should contain the following sort of output:

```
Driver script for phase_find_matches job flex_search
Current time: Tue Jan 17 14:16:20 2012
Number of CPUs requested = 1
.
```

```
.
.
Combining subjob log files . . .

*** Contents of flex_search_sub_0.log ***
.
.
.
phase_find_matches: Tue Jan 17 14:16:21 2012

Source = <TPATH>/flexDB.phdb
Hypothesis = DHHRR_9
Job name = flex_search
Options:
-NOCHECKPOINT
-flex
-subjob flex_search_sub_0
-subproc
Assigning default positional tolerances of 2.00
Applying QSAR model in DHHRR_9.qsar
Applying excluded volumes in DHHRR_9.xvol

Number of records to search = 100
.
.
.
ID,#Confs,#Matches,#Searched
mol_1,27,33,1
mol_3,80,0,3
mol_4,6,0,4
.
.
.
mol_97,49,0,97
mol_98,72,0,98
mol_99,2,0,99

Writing 3 hits to flex_search_sub_0-hits.list

*******************************************************************************

Finished processing the current batch of 100 records

Cumulative totals:
Number of molecules searched = 100
Number of conformers searched = 3625
Number of matches found = 110
Number of molecules that produced matches = 3
Number of hits written = 3

*******************************************************************************
.
.
.
*** End of flex_search_sub_0.log ***

Combining sorted hits from completed subjobs
```

```
Importing hits from flex_search_sub_0-hits.list . . .
Number of hits stored so far = 3

Writing 3 sorted hits to flex_search-hits.maegz . . .

Hits from completed subjobs successfully combined

All subjobs successfully completed
CPU time = 0.01 sec

phase_find_matches successfully completed

Driver script for phase_find_matches job flex_search finished
Current time: Tue Jan 17 14:17:08 2012
Elapsed time = 00:00:48
phase_find_matches results for flex_search are complete
```

Since the conformer generation options and the seed structures (see comments at the end of Section 1.6) in this job are identical to those used when `phase_database` was run to create conformers for the first 100 molecules in the standard database, so we would expect the same results for the `flex_search` and `basic_search` jobs with respect to the first 100 molecules. Verify this by examining the files `flex_search.log` and `basic_search_dbsearch.log`. You should observe identical numbers of matches and hits in these two files for `mol_1`, `mol_34`, and `mol_43`.

1.13 Using Site Masks

When the `phase_find_matches` *<minSites>* parameter is smaller than the number of sites in the hypothesis, partial matching is invoked. So in the case of a 5-point hypothesis, one could set *<minSites>* equal to 4 to indicate that the search should consider 4-point matches if no 5-point matches are found. This does not mean that every single 4-point match will always be identified; rather, it means that if a given molecule fails to produce any 5-point matches, the search will move on to consider 4-point matches. However, if a molecule produces at least one 5-point match, then there will be no subsequent search for 4-point matches. Note that this is the default partial matching behavior. One can force `phase_find_matches` to exhaustively consider all 5-point and all 4-point matches by supplying the `-ex` flag.

Continuing with the 5-point hypothesis example, any 4 points of the correct type which also satisfy the geometric matching criteria would constitute a partial match. However, suppose you know that a ligand cannot bind to a particular receptor unless it contains a positive site and an aromatic ring. In that case, you would be interested in 4-point matches only if they contain those pharmacophoric elements.

To enable this sort of searching you must create a *site mask* file. Consider the hypothesis we've been using, for which the site types are D, H, H, R, and R. Suppose we wanted to require that every partial match contain the donor site and the first aromatic site. The corresponding site mask file should then contain the following 5 lines:

```
4 D 1
5 H 0
6 H 0
9 R 1
11 R 0
```

The first two columns are identical to those in `DHHRR_9.xyz`, whereas the third column contains the mask value. A value of "1" indicates that the site must be matched, and "0" means that matching is optional. With this site mask and with *<minSites>* = 4, matches involving all 5 sites would first be sought. If no 5-point matches are found, the search would move onto 4-point matches, and the site mask would be applied.

In order to be recognized, the site mask file must be named *<hypoID>*`.mask`, where *<hypoID>* is the prefix used for all the other hypothesis-related files. Accordingly, while in the `userFiles` directory, use a text editor to create the file `DHHRR_9.mask`, type the above 5 lines into the file, using spaces (not tabs) to separate values, and save your changes.

While still in the `userFiles` directory, launch the database searching job as follows:

```
Shell> $SCHRODINGER/phase_find_matches $TPATH/stdDB.phdb DHHRR_9 site_mask \
       -match 4 -NOCHECKPOINT
```

When the job finishes, examine the file `site_mask.log`. You should find that a total of 3182 matches were identified, and a total of 70 hits were written to `site_mask-hits.maegz`. In the original `basic_search` job, where partial matching wasn't used, there were only 340 total matches and 28 hits. So by requiring only 4 sites to match, many additional matches and hits were found.

To verify that every hit in `site_mask-hits.maegz` matches both the donor site and the first aromatic site, issue the following command:

```
Shell> gunzip -c site_mask-hits.maegz | grep "D("
```

This should produce the output:

```
  "D(5) H(-) H(6) R(10) R(9)"
  "D(3) H(5) H(-) R(10) R(11)"
  "D(5) H(8) H(9) R(10) R(-)"
.
.
.
  "D(4) H(7) H(6) R(9) R(-)"
  "D(7) H(-) H(11) R(15) R(16)"
  "D(10) H(12) H(15) R(18) R(-)"
```

As discussed in Section 1.10, these Matched Ligand Sites strings indicate how the hit matched the hypothesis. For example, "`D(5) H(-) H(6) R(10) R(9)`" indicates that the 5th site in the hit, a donor, was matched to the first site in the hypothesis, the 6th site in the hit, a hydrophobe,

was matched to the 3rd site in the hypotheses, etc. When a site in the hypothesis is not matched by the hit, a hyphen appears, e.g., `H(-)`. Note that you will see no hyphens for the donor site or the first aromatic site because the site mask required that both of these be matched.

1.14 Using Feature-Matching Rules

Suppose that you wish to allow a hydrophobic site in the hypothesis to match either a hydrophobic or aromatic site in the database you are searching. Suppose further that if the hydrophobic site is not matched to one of these two types of sites, you never want it to inadvertently match an ionizable site. These sorts of conditions may be imposed by defining *feature-matching rules*.

A feature-matching rule for a given site in the hypothesis consists of a string of *permitted features*, and an optional string of *prohibited features*. As the names suggest, permitted features define what a given site is allowed to match, while prohibited features indicate what that site is *not allowed* to match. A rule must be created for each site in the hypothesis, following the same order as in `<hypoID>.xyz`.

Feature-matching rules are best explained by way of example, so consider the following set of rules for the 5-point hypothesis we have been using:

```
4 D
5 HR NP
6 H
9 R
11 RH
```

The first rule "`4 D`" is just the default way of matching, i.e., donor site 4 is permitted to match only another donor site. However, the second rule "`5 HR NP`" indicates that the hydrophobic site 5 is permitted to match either a hydrophobic or an aromatic site, but if it does not match either of these, then it *cannot* match a negative ionizable or positive ionizable site.

Note that the first character in each permitted feature string must be the actual site type that appears in `DHHRR_9.xyz`, (so, e.g., the last rule is "`11 RH`" rather than "`11 HR`") and none of the permitted features may appear in the prohibited feature string. Also note that prohibited features are only relevant in the case of a partial match.

When a given site in the hypothesis is allowed to match more than one type of site in the database (e.g., the 2nd and 5th rules above), this condition is described as using *mixed permitted features*. An important consequence of doing this is that all vector scoring will be turned off, so that matches between vector and non-vector features (e.g., H↔R) aren't automatically scored lower than matches between two vector features.

> *In previous releases, performing a find+fetch search with mixed permitted features had another important consequence: the pre-screen using 3D keys was not performed because it required a potentially large number of separate SQL queries in order to consider all possible*

combinations of permitted features. This could become quite expensive, especially when partial matching was used, because a separate SQL query was already performed for each unique subset of `<minSites>` *sites in the hypothesis. The new database searching tools perform only one SQL query for each type of pre-filter, so the speed of the pre-screen is essentially independent of the complexity of the hypothesis matching rules.*

Feature-matching rules must be stored in a file named `<hypoID>.rules`, so while residing in the `userFiles` directory, use a text editor to create `DHHRR_9.rules`, type the above rules into that file, using spaces to separate values, and save the changes.

Launch the database searching job as follows:

```
Shell> $SCHRODINGER/phase_find_matches $TPATH/stdDB.phdb DHHRR_9 feature_rules \
       -match 4 -NOCHECKPOINT
```

Be aware that the site mask created in the previous section is also being applied, so the donor site and the first aromatic site must be matched. There are no logical inconsistencies in combining this site mask and the current feature rules. However, it would be inconsistent if the site mask required the first hydrophobic site to be matched, because we have defined prohibited features for that site, which are applied only if the site is not matched to a permitted feature.

When the job finishes, examine the file `feature_rules.log`. You will see that, compared to the previous `site_mask` search, the total number of matches has increased from 3182 to 6072, and the number of hits has increased from 70 to 88. This is because the use of mixed permitted features leads to a greater number of ways to match the hypothesis to a given molecule.

To examine the various mappings in the hit file, issue the following command:

```
Shell> gunzip -c feature_rules-hits.maegz | grep "D("
```

This should produce the output:

```
  "D(5) H(-) H(6) R(10) R(9)"
  "D(6) H(8) H(9) R(10) R(12)"
  "D(5) H(6) H(7) R(10) R(12)"
.
.
.
  "D(4) R(12) H(-) R(14) R(13)"
  "D(7) R(13) H(9) R(12) R(-)"
  "D(7) H(-) H(8) R(10) H(9)"
```

Observe that among the 88 hits, there are many instances of the first hydrophobic site in the hypothesis being matched to an aromatic ring, e.g., `"D(8) **R(11)** H(-) R(12) R(13)"`, and the second aromatic ring in the hypothesis being matched to a hydrophobic site, e.g., `"D(4) H(6) H(-) R(9) **H(5)"**.`

Verifying that the prohibited rules have been enforced is not so straightforward, but it's possible to do so by visual inspection of the hits that fail to match the first hydrophobic site.

In order to avoid inadvertent application of feature-matching rules in subsequent sections, remove `DHHRR_9.rules` before proceeding:

```
Shell> rm DHHRR_9.rules
```

1.15 Using Feature-Specific Positional Tolerances

As noted in Section 1.9, database matches are found by comparing intersite distances from the hypothesis with those of the database conformers. A single tolerance *<deltaDist>* is applied to all intersite distances, and a single positional tolerance is applied to all matching sites after alignment, which means that different types of pharmacophore features are treated equivalently. Because certain types of ligand-receptor interactions are stronger and more specific, it often makes sense to define different positional tolerances for matching different types of features. To enable this, you must create a file containing *feature-specific positional tolerances*.

Note that *<deltaDist>* is still applied to identify all the initial matches, so the value of this parameter may be adjusted independently of the positional tolerances. While the mathematical relationship between intersite distance matches and positional matches is quite complex for hypotheses with several sites, choosing a value of *<deltaDist>* that is equal to the sum of the two largest positional tolerances will ensure that *<deltaDist>* is no more stringent than the positional tolerances.[2] Be aware that if you supply a set of positional tolerances and you do not specify a value for *<deltaDist>*, this rule is applied to assign a default value. Because this often leads to a fairly large *<deltaDist>* (e.g., 3 or 4 Å), the number of intersite distance matches may be excessive, and the computational cost of searching may increase significantly. In most cases, using such a large *<deltaDist>* is not worth the extra cost because the vast majority of the additional matches will provide poor overall fits to the hypothesis..

To be recognized, the feature-specific tolerances file must be named *<hypoID>*`.tol`. While in the `userFiles` directory, use a text editor to create the file `DHHRR_9.tol` and type in the following 9 lines of text, using spaces to separate the letters from the numbers:

```
A 1.50
D 1.50
H 2.00
N 0.75
P 0.75
R 2.00
X 1.50
Y 1.50
Z 1.50
```

[2] If the hypothesis contains two or more sites of a given type, and the positional tolerance assigned to that type is the largest, one would simply double that tolerance to obtain a *<deltaDist>* which is no more stringent than the positional tolerances.

Note that we are defining a tolerance in Å for every possible feature type (including custom features X, Y, Z). Strictly speaking, we really need only define tolerances for D, H and R in order to search with the current hypothesis, but you should get into the habit of considering all possible features because you may want to use a single set of tolerances for all of your database searching. Be aware that if you accidentally omit a feature type that's contained in your hypothesis, a default tolerance of 1.0 Å will be used.

It should be apparent that this mechanism does not allow you to assign different tolerances to different instances of the same feature type. So with our current hypothesis, the same tolerance would be applied to both hydrophobic sites, and the same tolerance would be applied to both aromatic sites. Section 1.16 describes how to define site-specific tolerances that allow you to overcome this restriction.

Save the file `DHHRR_9.tol` and launch the job as follows:

```
Shell> $SCHRODINGER/phase_find_matches $TPATH/stdDB.phdb DHHRR_9 feature_tol \
       -d 2.0 -NOCHECKPOINT
```

We have not used the option `-match <minSites>`, so all 5 sites are required to match and the existing site mask will have no effect. We are overriding the default `<deltaDist>` value, which would have been 4.0 since the two largest positional tolerances are 2.0.

When the job finishes, examine the file `feature_tol.log`. You will see that a total of 278 matches were found, and a total of 27 hits were written to `feature_tol-hits.maegz`. This is a reduction from the 340 matches and 28 hits found when positional tolerances were not used.

In order to avoid inadvertent application of these positional tolerances in subsequent sections, remove `DHHRR_9.tol` before proceeding:

```
Shell> rm DHHRR_9.tol
```

1.16 Using Site-Specific Positional Tolerances

When using feature-specific tolerances, as described in Section 1.15, it is not possible to distinguish between different instances of the same feature type. So if your hypothesis contains two hydrophobic sites, both would be matched with the same tolerance. However, there may be cases where you will want to use different tolerances, and we describe how to do that here.

Site-specific tolerances must be stored in a file named `<hypoID>.dxyz`, so use a text editor to create the file `DHHRR_9.dxyz` and type in the following 5 lines of text, using spaces to separate the values on each line:

```
4 D 1.50
5 H 2.00
6 H 1.00
```

```
9 R 2.00
11 R 1.00
```

Observe that we are placing tighter tolerances on matching the second hydrophobic site and the second aromatic site. After saving the file, launch the job as follows:

```
Shell> $SCHRODINGER/phase_find_matches $TPATH/stdDB.phdb DHHRR_9 site_tol \
       -d 2.0 -NOCHECKPOINT
```

When the job finishes, examine the file `site_tol.log`. You will see that a total of 125 matches were found and a total of 24 hits were written to `site_tol-hits.maegz`. This is a reduction from the 278 matches and 27 hits found when positional tolerances of 2.0 were applied to both hydrophobic sites and both aromatic sites.

In order to avoid inadvertent application of site-specific matching tolerances in subsequent sections, remove `DHHRR_9.dxyz` file before proceeding:

```
Shell> rm DHHRR_9.dxyz
```

1.17 Working with Database Subsets

There may be instances where a user wishes to perform operations on only a subset of a Phase database. For example, suppose you wanted to restrict a database search to the set of molecules that produced hits in a previous search. Or perhaps you have added new molecules to a database that already contains conformers and sites, and you wish to generate conformers and sites for only the new molecules. Or maybe you want to create a database subset by performing a logical operation (AND, OR, NOT) on two existing subsets. In this section, we demonstrate how to accomplish these sorts of tasks using `phase_database`.

To see the available options, type the following:

```
Shell> $SCHRODINGER/phase_database –help_subset
```

Observe that you may create subsets from a hit file, from conformer and site queries, from a file containing a list of titles, and from a logical combination of existing subsets.

We shall first demonstrate how to create a subset from a hit file and then show how to restrict a database search to that subset. Ensure that you are in the `userFiles` directory and that the file `basic_search-hits.maegz` is present. With the `TPATH` environment variable defined as described in Section 1.4, issue the following command to create a subset from the aforementioned hit file:

```
Shell> $ $SCHRODINGER/phase_database $TPATH/stdDB.phdb subset \
       -hits basic_search-hits.maegz -osub myHits -NOJOBID
```

We are using the -NOJOBID flag so that the task runs as a foreground process rather than as a Schrödinger job. When the prompt returns, the current directory should contain the subset file myHits_phase.inp, which holds LIGAND_NAME records for the 28 molecules in basic_search-hits.maegz:

```
LIGAND_DIR = database_ligands
LIGAND_NAME = block_1/mol_1
LIGAND_NAME = block_1/mol_34
.
.
.
LIGAND_NAME = block_1/mol_125
LIGAND_NAME = block_1/mol_126
```

Note that Phase subset files always have a name of the form *<subset>*_phase.inp, and that they normally contain a LIGAND_DIR record in addition to the LIGAND_NAME records, although the LIGAND_DIR record is optional.

Now suppose you wanted to repeat the search that identified those 28 hits, but you wanted to use a tighter intersite distance matching tolerance. In that case, there is no need to search the entire database, because molecules that did not match the hypothesis in the initial search cannot possibly match it if the tolerance is reduced. Therefore, you can restrict the search to the 28 molecules in the subset you just created. Accordingly, launch the following database searching job:

```
Shell> $SCHRODINGER/phase_find_matches $TPATH/stdDB.phdb DHHRR_9 tight_match \
       -isub myHits -d 1.3 -NOCHECKPOINT
```

When the job finishes, examine the file tight_match.log. You should find that only 17 of the 28 molecules satisfied the stricter distance matching tolerance.

Suppose you wanted to determine which of the 28 molecules *failed* to satisfy the new distance-matching tolerance. First, create a subset from the 17-molecule hit file as follows:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb subset \
       -hits tight_match-hits.maegz -osub tightHits -NOJOBID
```

Now use the logical NOT operator to subtract the 20-member subset from the original 27-member subset:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb subset \
       -logic NOT -isub1 myHits -isub2 tightHits -osub missedHits -NOJOBID
```

The file missedHits_phase.inp should now contain LIGAND_NAME records for only the molecules that failed to satisfy the 1.3 Å tolerance, i.e.,

```
LIGAND_DIR = database_ligands
LIGAND_NAME = block_1/mol_1
LIGAND_NAME = block_1/mol_34
```

```
LIGAND_NAME = block_1/mol_43
LIGAND_NAME = block_1/mol_118
LIGAND_NAME = block_1/mol_119
LIGAND_NAME = block_1/mol_120
LIGAND_NAME = block_1/mol_121
LIGAND_NAME = block_1/mol_123
LIGAND_NAME = block_1/mol_124
LIGAND_NAME = block_1/mol_125
LIGAND_NAME = block_1/mol_126
```

As a final example, we illustrate how subsets are used to restrict conformer and site creation to a portion of a database. Issue the following command to add the single-conformer molecules in `$TPATH/userFiles/newMol.mae.gz` to the database you created in Sections 1.4-1.7:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb import \
       -i $TPATH/userFiles/newMol.mae.gz -verbose
```

This operation will require about one second to finish. The tail end of the file `database_import.log` should look like:

```
.
.
.
LIGAND_NAME = block_1/mol_149  # Title = "serot-23"  Confs = 1
LIGAND_NAME = block_1/mol_150  # Title = "serot-24"  Confs = 1
LIGAND_NAME = block_1/mol_151  # Title = "serot-25"  Confs = 1
Committing 25 records
Updating top-level database files . . .

A total of 25 records added
Total number of records stored = 151

Removing restart directory

CPU time = 0.82 sec

phase_database successfully completed

Driver script for phase_database job database_import finished
Current time: Wed Jan 18 14:46:53 2012
Elapsed time = 00:00:01
phase_database results for database_import are complete
```

At this point, the database contains 126 molecules with multiple conformers and pharmacophore sites, and 25 molecules with single conformers and no sites. To bring this database up-to-date, we need to generate conformers and sites for only the last 25 molecules. Observe that when you import structures, `phase_database` always creates a subset file, *<jobName>*`_new_phase.inp`, with the newly created record numbers. We used the default job name for the last import, so the records we need are contained in the file `database_import_new_phase.inp`.

To create sites and conformers for only the 25 newly added molecules, launch a `phase_database` revise job as follows:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb revise new_mol \
       -confs all -isub database_import_new
```

When the job finishes, you should see the following sort of output at the end of the file `new_mol.log`:

```
.
.
.
All subjobs successfully completed.  Updating top-level database files . . .
Revision of 25 out of 25 records completed
Average number of conformers per revised record = 10.68
Total number of records stored = 151

Removing restart directory

CPU time = 0.05 sec

phase_database successfully completed

Driver script for phase_database job new_mol finished
Current time: Wed Jan 18 14:59:52 2012
Elapsed time = 00:00:08
phase_database results for new_mol are complete
```

All 151 molecules in the database now have multiple conformers and pharmacophore sites.

1.18 Working with Database Properties

The Maestro and SD files that you import to create a Phase database usually contain any number of properties for each molecule, and these properties are stored within the block database files under the `database_ligands/` directory. You may also generate Canvas properties when you run a revise job with the `-props` flag, and these are also stored in the block database files. The `phase_database` "extract" task may be used to extract all such properties to a separate SQLite database that can be searched using SQL queries.

We will be extracting properties from the database `$TPATH/stdDB.phdb`, so you need to complete Sections 1.4-1.7 and Section 1.17 in order to obtain the results presented in this section.

To see the available options for the extract task, type the following:

```
Shell> $SCHRODINGER/phase_database -help_extract
```

Although we will not be using the `-append` flag in this tutorial, it's important to be aware of it. If you create a large database, extract properties, then import a few new structures to the database, you can extract properties for just the newly added records by using `-append`. This is much faster than extracting properties for the entire database. Note, however, that once the extraction has been done for the new records, use of the `-append` flag in a subsequent extraction will not

re-extract properties for the newly added records. For this reason, if you intent to run a revise job to create conformers and sites for the newly added records, you should extract properties *after* the revise job. This will guarantee that any changes which take place as a result of the revise job (e.g., `num_confs`, `has_sites`), will be reflected in the extracted properties. If you inadvertently do the extraction before running the revise job, you will need to re-extract properties for the entire database in order to get up-to-date properties.

While in the `userFiles/` directory, issue the following command to launch the property extraction job:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb extract -map -stats
```

The job should require only a second or two to finish, after which you will find the following new files in the directory `$TPATH/stdDB.phdb`:

File Name	Description
`database.sqlite`	SQLite database with all extracted properties.
`database_props.csv`	Comma-separated file with a copy of all the properties in `database_props.sqlite`.
`database_props_map.csv`	Contains mappings between the SQLite property names and the source property names in the block database files. Created only if the `-map` flag is used.
`database_props_stats.csv`	Property statistics: `count`, `min`, `max`, `avg`, where `count` is the number of records for which the property exists. Created only if the `-stats` flag is used.

The file `database_props_map.csv` is particularly useful because it provides a vertical list of all the properties that can be used in SQL queries. In cases where `SQLiteProp` and `SourceProp` differ, the former should be used when performing a query. Be aware that for imported properties with a conformational dependence (e.g., `r_mmod_Potential_Energy-MMFF94s`), only the value for the very first conformer is stored in the block database files and in the SQLite property database.

Now that `$TPATH/stdDB.phdb/database.sqlite` has been created, we can pose a SQL query to find records that satisfy some property filter. This is done by running a `phase_database` "query" job. To see the available options, type the following

```
Shell> $SCHRODINGER/phase_database –help_query
```

Observe that the query may involve records that satisfy a particular condition, records that contain specific values of a given property, or records that match a SMARTS string. In the case of SMARTS, it's not actually necessary to extract properties before performing the query.

Suppose you wanted to identify molecules with a value of `r_user_Activity` greater than 7.5. To accomplish this, issue the following command:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb query active \
```

```
    -where "r_user_Activity > 7.5"
```

The query must be enclosed in single or double quotes so that it is interpreted as a single argument by the shell. When the job finishes, the files `active_matches.csv` and `active_matches_phase.inp` will be created in the current directory. Use a text editor or spreadsheet program to examine `active_matches.csv` and verify that there are 13 matching records, and `r_user_Activity` is greater than 7.5 in each case. Verify that the subset file `active_matches_phase.inp` contains the same 13 records.

The following are examples of other valid `-where` queries:

```
-where "num_confs = 1"
-where "title <> 'endo-1'"
-where "r_user_Activity > 7.5 OR num_confs > 10"
-where "r_user_Activity BETWEEN 7.0 AND 8.0"
-where "title LIKE 'serot%'"
```

Note that in some cases single quotes must be used within the double quotes to ensure that the query is correctly interpreted by the SQLite engine.

1.19 Converting Databases

Databases created with previous releases of Phase using `phasedb_manage`/`phasedb_confsites` (or using the Phase GUI) are not compatible with the tools described in this tutorial, so you will need to convert those databases to the new format in order to search them or to make modifications to them.

A database may be converted to the new format, or merged into an existing database, by running a `phase_database` "convert" job. To see the available options, type the following:

```
Shell> $SCHRODINGER/phase_database -help_convert
```

The destination database is the usual *<path/name>* argument we have been specifying with every `phase_database` command, whereas the *source* database is the one that is being converted/merged, and it can be stored in the old or new format.

Ensure that the `TPATH` environment variable is defined as described in Section 1.4, and move to that directory:

```
Shell> cd $TPATH
```

The tutorial comes with a compressed tar archive that contains a small database created using Phase 3.3. Extract that archive as follows:

```
Shell> gunzip -c userFiles/Phase_3.3_DB.tar.gz | tar xf -
```

Now list the contents of the extracted database directory:

```
Shell> ls -F Phase_3.3_DB
```

You should see the following files:

```
oldDB_dbInfo.log  oldDB_feature.ini  oldDB_master_phase.inp
oldDB_dbversion   oldDB_ligands/     oldDB_phasedb
```

Submit the conversion job as follows:

```
Shell> $SCHRODINGER/phase_database $TPATH/Phase_3.4_DB.phdb convert old_to_new \
       -source $TPATH/Phase_3.3_DB/oldDB -new
```

As with other `phase_database` jobs, it is possible to launch a conversion job from a local host that does not have physical access to the source or destination databases, but the remote host must have access to both databases via the paths supplied in the launch command.

Since only 36 records are being converted, this job should finish in less than a minute. When the conversion is complete, you should see the following sort of output at the end of the file `old_to_new.log`:

```
.
.
.
Converting block_1/mol_34 to block_1/mol_34 (34 of 36)
Converting block_1/mol_35 to block_1/mol_35 (35 of 36)
Converting block_1/mol_36 to block_1/mol_36 (36 of 36)
Committing 36 records

********************************************************************************

All records assigned to old_to_new_sub_0 have been processed
CPU time = 7.45 sec

phase_database successfully completed

*** End of old_to_new_sub_0.log ***

All subjobs successfully completed.  Updating top-level database files . . .
Conversion of 36 out of 36 records completed
Average number of conformers per converted record = 28.81
Total number of records stored = 36

Removing restart directory

CPU time = 0.78 sec

phase_database successfully completed

Driver script for phase_database job old_to_new finished
Current time: Fri Jan 20 13:56:39 2012
Elapsed time = 00:00:10
phase_database results for old_to_new are complete
```

Now list the contents of the destination database directory:

```
Shell> ls -F Phase_3.4_DB.phdb
```

You should see the following files:

```
database.sqlite       database_info.log          database_summary.csv
database_dbversion    database_ligands/
database_feature.ini  database_master_phase.inp
```

Verify that there are 36 `LIGAND_NAME` records in the file `database_master_phase.inp` and that the directory `database_ligands/block1` contains the files `block_struct_1.sqlite`.

To compare the sizes of the source and destination databases, issue the following commands:

```
Shell> du -k Phase_3.3_DB
Shell> du -k Phase_3.4_DB.phdb
```

You should see that the source database consumes 1104 blocks (1 kB per block), whereas the destination database consumes only 388 blocks, which is a reduction of 65%.[3] In most cases, databases stored in the new format are only 35-45% as large as those stored in the old format.

At this point you may safely delete the source database as follows:

```
Shell> rm –rf Phase_3.3_DB
```

1.20 Exporting Structures from a Database

At some point you may wish to export structures from a Phase database and write those structures to Maestro or SD files. In this section we demonstrate how this may be done by running a `phase_database` "export" job.

We will be exporting structures from the database `$TPATH/stdDB.phdb`, so you need to complete Sections 1.4-1.7 and Section 1.17 in order to obtain the results presented in this section.

To view the available options type the following:

```
Shell> $SCHRODINGER/phase_database help_export
```

Observe that if running under Schrödinger job control, the *<base>* parameter must contain an absolute path that exists on the host where the job will run.

For large databases, the exported structures may require a great deal of disk space, so `phase_database` has an option to impose a quota on the total disk space consumed by the files it

[3] The database sizes you observe may differ slightly from what is reported here.

creates. The program will abort as soon as that quota is reached. So if there are, say, 10 Gbytes of available disk space on the filesystem that you will be exporting to, it's a good idea to set the quota to something smaller than 10 Gbytes, particularly if the filesystem is shared with other users.

Ensure that the `TPATH` environment variable is defined as described in Section 1.4, and move to the directory `$TPATH/userFiles`:

```
Shell> cd $TPATH/userFiles
```

Issue the following command to export the first conformer only for each molecule in `$TPATH/stdDB.phdb`, and write those structures to a series of SD files, with a maximum of 100 structures per file:

```
Shell> $SCHRODINGER/phase_database $TPATH/stdDB.phdb export \
       -osd $TPATH/userFiles/stdDB_export -get 1 -limit 100
```

The export should only take a second or two, after which you should find the files `stdDB_export_1.sdfgz` and `stdDB_export_2.sdfgz` in the current directory, with 100 and 51 structures, respectively.

1.21 Running on Multiple Processors

`phase_database` "revise" and "convert" tasks and `phase_find_matches` may be run as multiple-CPU jobs on an appropriately configured cluster, a multiprocessor workstation, or across a series of networked hosts. Because a Phase database is held in a fixed, central location, (i.e., not copied between hosts at runtime), these jobs are not intended to be run in a grid computing environment.

It is beyond the scope of this tutorial to provide step-by-step instructions for setting up an environment to allow multiple-CPU jobs, so it is assumed that the user is already able to run other Schrödinger software on multiple CPUs. For further details, please consult the Product Installation Guide.

Aside from configuration issues, the first requirement is that the database be located in a directory that is uniformly accessible to all machines that will need access to the database. This means the absolute path to the database must be identical on all the applicable CPUs. Further, `phase_database` jobs must operate on multiple blocks in order to use multiple CPUs. For example, if you wished to split a `phase_database` revise job across 4 CPUs, you would have to be creating sites/conformers/properties for at least 4 database blocks. If you wished to split a `phase_database` convert job across 4 CPUs, the converted records would have to be spread over at least 4 blocks in the destination database.

Assuming these requirements are met, a `phase_database` or `phase_find_matches` job may be split across multiple processors by using the option `-HOST` option to indicate how many CPUs should be used.

Suppose the file `$SCHRODINGER/schrodinger.hosts` contains a host entry named `hostX` with16 processors, and you wish to split a `phase_database` convert job across 4 of those processors. Accordingly, you could issue the following sort of command to launch the job:

```
Shell> $SCHRODINGER/phase_database -db <path/name> revise multi_cpu \
       -confs auto -HOST hostX:4
```

Similarly, if you wanted to run a `phase_database` convert job or a `phase_find_matches` job on 4 processors, you could use commands like:

```
Shell> SCHRODINGER/phase_database <path/name> convert multi_cpu \
       -source <source> -new -HOST hostX:4

Shell> $SCHRODINGER/phase_find_matches <path/name> <hypoID> multi_cpu \
       -HOST hostX:4 -NOCHECKPOINT
```

If you do not currently have access to a machine with multiple processors, you can still observe how jobs are split up by mimicking a request for 4 processors on the *localhost*. For example:

```
Shell> $SCHRODINGER/phase_find_matches <path/name> <hypoID> multi_cpu \
       -HOST localhost:4 -NOCHECKPOINT
```

This command will result in the launching of 4 subjobs that will run as concurrent processes on the machine you are logged into. The behavior will be much the same as if you were actually running on 4 independent CPUs, except that it won't be any faster than a regular single-processor job.

If you wish to split a job across more than one host on the network, then you must enclose the list of hosts in double quotes and separate them with spaces. For example:

```
Shell> $SCHRODINGER/phase_find_matches <path/name> <hypoID> multi_host \
       -HOST "hostX:4 hostY:2 hostZ:8" -NOCHECKPOINT
```

In this example, a total of 14 subjobs would be launched.