Using AutoDock 4 for Virtual Screening

Written by William Lindstrom, Garrett M. Morris,

Christoph Weber and Ruth Huey

The Scripps Research Institute
Molecular Graphics Laboratory
10550 N. Torrey Pines Rd.
La Jolla, California 92037-1000
USA

29 January 2008, v2

Contents

Introduction

This tutorial will introduce you to the process of virtual screening using UNIX shell commands and python scripts in the **AutoDock** suite of programs. There are nine steps in the tutorial in which we will prepare a library of ligand files and corresponding AutoGrid and AutoDock parameter files for the library, use AutoGrid to calculate maps, launch AutoDock calculations for each ligand (see Figure 1.1 below) followed by two analysis steps in which we will extract and evaluate the results (Exercises 10 and 11, not shown). In addition, we will focus on the data structures and documentation necessary for large scale calculations.
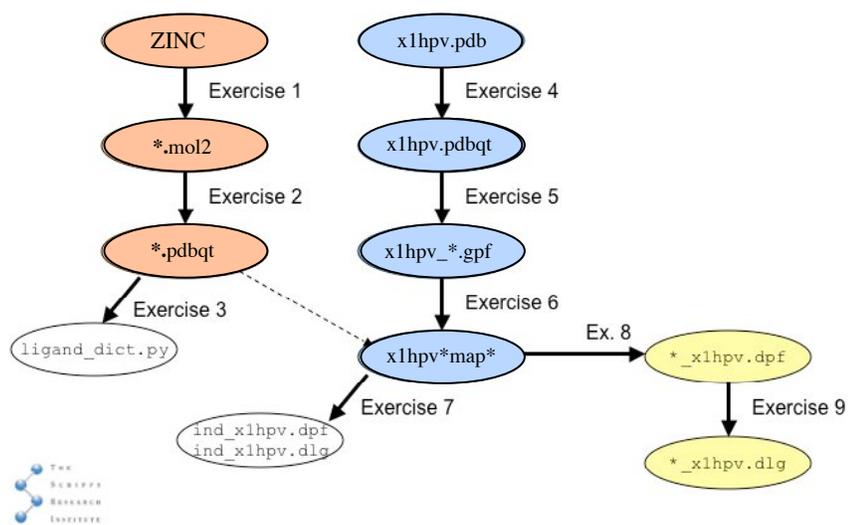
Figure 1.1 VS Tutorial Map

Before We Start…

We'll use the directory /usr/tmp for the tutorial today. In practice you'll use a directory of your own choosing.

Open a Terminal window and then type this at the UNIX, Mac OS X or Linux prompt:

Type this: ➡

```
cd /var/tmp
mkdir tutorial
cd tutorial
pwd
```

tutorial

We will represent directories as shaded boxes connected with lines to illustrate the data structure built in these exercises. This box represents the 'tutorial' directory you have just created.

Set up for today's exercises by checking out the VSTutorial files from CVS, Concurrent Versions System. First setup access to CVS:

Type this: ➡

```
setenv CVSROOT :pserver:anonymous@mgl1.scripps.edu:/opt/cvs
echo $CVSROOT
cvs login
```
(When asked for a password, just press **return**.)

Next, in the **/usr/tmp/tutorial** directory on the computer you are using here in the training room, check out the tutorial:

Type this: ➡

```
cvs co VSTutorial
cd VSTutorial
```

Figure 1.2 CVS Data

tutorial
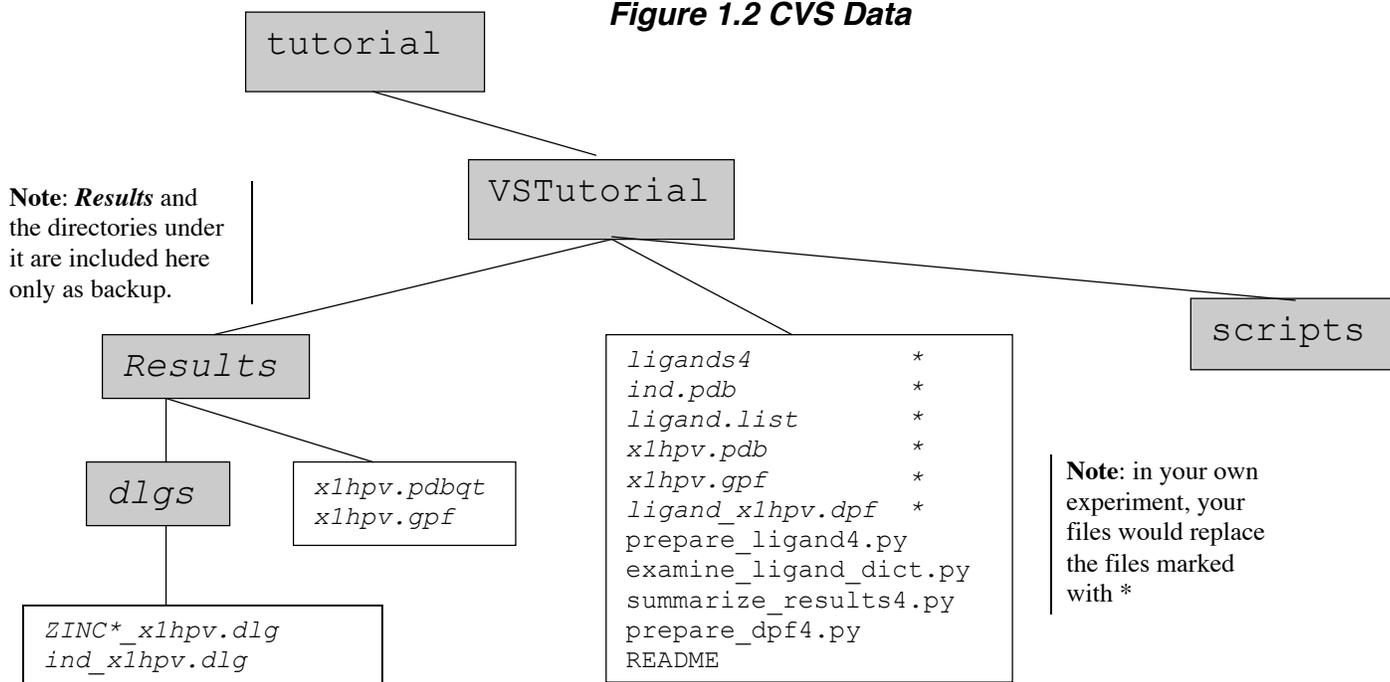
VSTutorial

Note: *Results* and the directories under it are included here only as backup.

Results

scripts

```
ligands4          *
ind.pdb           *
ligand.list       *
x1hpv.pdb         *
x1hpv.gpf         *
ligand_x1hpv.dpf  *
prepare_ligand4.py
examine_ligand_dict.py
summarize_results4.py
prepare_dpf4.py
README
```

dlgs

```
x1hpv.pdbqt
x1hpv.gpf
```

Note: in your own experiment, your files would replace the files marked with *

```
ZINC*_x1hpv.dlg
ind_x1hpv.dlg
```

$VSTROOT:
a short cut to the directory in
which your Virtual Screening
Tutorial activities will take
place.

Note: here we use the
backward-slanted single
quotation mark. UNIX
replaces strings enclosed by
this character by the result of
executing them.
Here `pwd` is replaced by
'/usr/tmp/tutorial/VSTutorial'
before **setenv** is executed.

```csh
#!/bin/csh
#
# $Id: ex00.csh,v 1.3 2005/01/31 18:11:28 lindy Exp $
#

# Because this script uses "pwd" to set VSTROOT it matters
# where (which directory) you run it from. This script
# should be run as "source ./scripts/ex00.csh" So,
# after you did your "cvs co VSTutorial" a "VSTutorial"
# directory was created and that's the one that should be
# your working directory when you source this script.
#
# Set up the root directory of the Virtual Screening
# Tutorial
#
setenv VSTROOT `pwd`
```

Type this: ⇨
```
source scripts/ex00.csh
echo $VSTROOT
```

tutorial

VSTutorial   **$VSTROOT**

Results   scripts
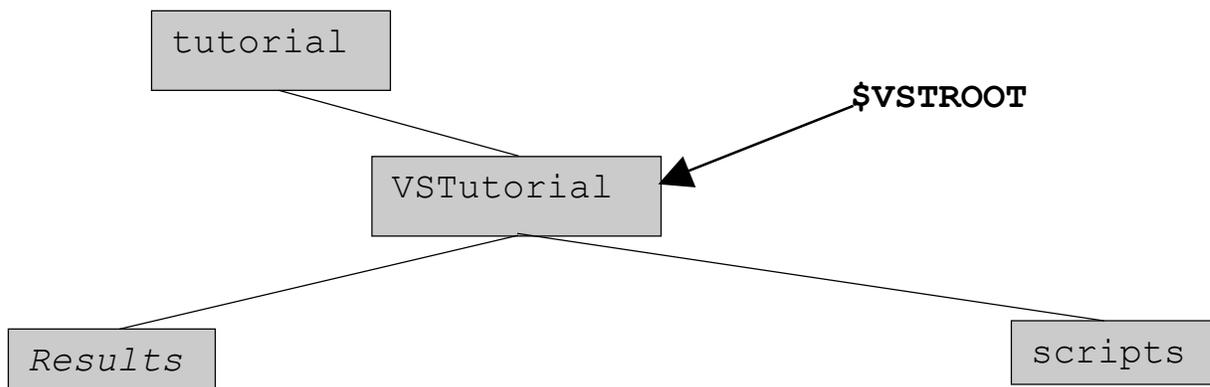
Figure 1.3 $VSTROOT

You must setup your environment to access to **python**, **adt**,
autodock4 and **autogrid4**.

For TSRI users:
```
source scripts/setpath4.csh
```

Type this: ⇨

Note: you will need to source this
script in any new terminal you
open during this tutorial to
properly set up the environment
in that new terminal.

(others need to edit the script for their local file systems)

FAQ – Frequently Asked Questions

1. *What library should I use for screening?*

   If you want to try and find **novel** compounds, you probably want to use a library designed for **diversity**, one which probes a large 'chemical space.' If there are small molecules which are known to bind to your macromolecule, you may want to construct a tailored library of related compounds.

2. *How much computational time should be invested in each compound? How many dockings, how many evaluations?*

   It depends on your receptor and on the computational resources available to you. One recent successful AutoDock Virtual Screening used 100 dockings with 5,000,000 evaluations per docking per compound.

3. *How do I know which docking results are 'hits'?*

   When the results are sorted by lowest-energy, the compounds which bind as well as your positive control or better can be considered potential hits. (Remember to allow for the ~2.1 kcal/mol standard error of AutoDock). If you have no positive control, consider the compounds with the lowest energies as potential hits.

4. *What's the best way to analyze the results?*

   Sort them by lowest energy first, then use ADT to inspect the quality of the binding.

5. *Will I need to visualize the results with the best energies?*

   Generally it is wise to inspect the top 30 to 50 results. Some people advocate visually inspecting the top 100-400 hits.

6. *What should I look for when I visualize a docked compound?*

   The first thing to check is that the ligand is docking into some kind of pocket on the receptor. The second is that there is a chemical match between the atoms in the ligand and those in the receptor. For example, check that carbon atoms in the

ligand are near hydrophobic atoms in the receptor while nitrogens and oxygens in the ligand are near similar atoms in the binding pocket. Check for charge complementarity. Check whatever else you may know about your particular system: for instance, if you know that the enzymatic action of your protein involves a particular residue, examine how the ligand binds to that residue. In the case of HIV protease, good inhibitors bind in a mode which mimics the transition state.

7. Where can I get help?

    The AutoDock mailing list is a good place to start. Information about it and other AutoDock resources can be found on the AutoDock Web site:

    `http://autodock.scripps.edu`

Exercise One: Populating the Ligand Directory: obtaining mol2 files

The library used for a virtual screening experiment is a selected group of ligand files. Sources of libraries include Maybridge (www.maybridge.com), MDL Mentor, Available Chemicals (UW-Madison), NCI among many others. Libraries are characterized according to their uniqueness, diversity and drug-likeness which is based on Lipinsky's "Rule of Five" which consists of four criteria: molecular weight <500, logP <5, number of hydrogen bond donors<5 and number of hydrogen bond acceptors < 10.

The size of the library which can be screened depends on the available computational resources. Typically libraries number in the tens to hundreds of thousands of files. It is practically impossible to test exhaustively any large chemical database. Libraries are constructed to maximize the chances of obtaining good 'hits' by focusing on ligand diversity.

NCI Diversity Set
To expedite drug discovery, the National Cancer Institute maintains a resource of more than 140,000 synthetic chemicals and 80,000 natural products for which it can provide samples for high-through-put screening (HTS). The NCI Diversity Set is a collection of 1990 compounds selected to represent the structural diversity in the whole resource.

ZINC
ZINC **I**s **N**ot **C**ommerical is a free database of over 4.6 million commercially-available compounds for virtual screening (blaster.docking.org/zinc). The first exercise illustrates setting up a data structure and populating the Ligands directory with 115 mol2 files from ZINC.

Documentation
Documenting each step of a computational experiment in sufficient detail to be able to reproduce it is an essential requirement. **README** files are one common form of documentation. Important sections in a README file for computation experiments include: Project, Author, Date, Task, Data sources, Files in this directory, Output files, Running Scripts and other notes on the location of the executable and environmental settings.

In the "**Before We Start**…" section, you set up local copies of the input files and executable scripts we will use today.

Procedure:

1. In **ex01.csh**, we create a working directory called **VirtualScreening** and two subdirectories: one called **Ligands** where we will do all the preparation of the ligand files and a second called **etc** where we'll keep a few extra, useful files.

Next we populate the Ligands directory by splitting a multimolecule file from ZINC into 115 separate files.

Finally, we add a positive control, **ind.pdb**, to the list of ligands.

```
#!/bin/csh# Create the directory in which all your Virtual
# Screening Tutorial activities will take place:

cd $VSTROOT
mkdir VirtualScreening

# Create the Ligands and etc subdirectories:
cd VirtualScreening
mkdir Ligands
mkdir etc

#make the Ligands directory the current working directory
cd Ligands

# use the UNIX utility csplit to divide the multi-molecule mol2
# file into separate files
cat $VSTROOT/zinc.mol2|csplit —ftmp —n —ks —
'%^@.TRIPSO.MOLECULE%' '/^@.TRIPOS.MOLECULE/' '{* }'

#Rename the tmp file according to ZINC identifier
# Here is the outline of how we do this:
# 1. Extract ZINCn8 from the tmpNNNN file and set to variable
# 2. If the Zn8.mol2 file does not exit, rename tmpNNNN files

foreach f (tmp*)
echo $f
set zid = `grep ZINC $f`
if !(-e "$zid".mol2) then
set filename = "$zid".mol2
else foreach n (`seq —w 1 99`)
if !(-e "$zid"_"$n".mol2) then
set filename = "$zid"_"$n".mol2
break
endif
end
endif
mv —v $f $filename
end

# Copy positive control ind.pdb down into Ligands:
cp $VSTROOT/ind.pdb .

#  Create the list of ligands in the etc directory:
\ls *mol2 *.pdb > $VSTROOT/VirtualScreening/etc/ligand.list
```

Note: here we split a file with many molecules in mol2 format into separate files to be processed in next exercises.

Type this: ▭⟹ `source $VSTROOT/scripts/ex01.csh`

On Mac OS X: `source $VSTROOT/scripts/ex01_mac.csh`

```
tutorial
            VSTutorial          ← $VSTROOT
    Results    VirtualScreening              scripts
        etc    Ligands
   ligand.list   ZINC*.mol2
                 ind.pdb
```
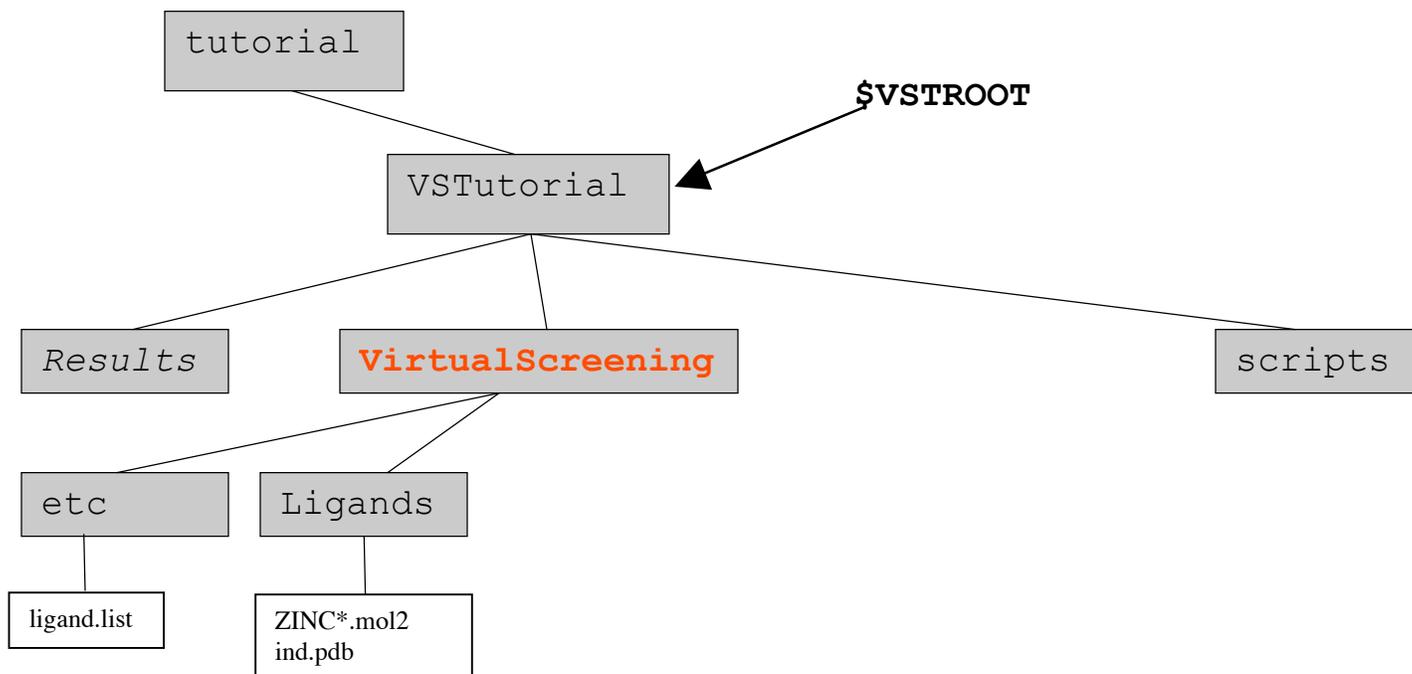
Figure 1.4 Exercise 1 Result

2. Let's look at the list we used
```
foreach f (`cat ../etc/ligand.list`)
    echo $f
    end
```

3. To confirm that the **foreach** loop did what we expected, list the mol2 files. Use **wc** (word count) for counting. Check that the number of mol2 files **115** plus **1** the number of pdb files, i.e. ind.pdb the positive control, matches the number of ligands in the ligand.list file **116.**

Type this: ▭⟹
```
\ls *.mol2 | wc —l
\ls *.pdb |wc -l
wc -l ../etc/ligand.list
```

4. Document the experiment:

Type this: ▭⟹
```
cd $VSTROOT
vim README
```

Fill in **Project**, **Author**, **Date**, **Data sources**, **Files in this directory** and an entry for this section's procedure.

Exercise Two: Processing the ligands: mol2 to pdbqt.

An AutoDock experiment results in docked ligand structures which represent the best (lowest energy) conformation found in the specified search space. Input molecule files for an AutoDock experiment must conform to the set of atom types supported by AutoDock. This set consists of united-atom aliphatic carbons, aromatic carbons in cycles, polar hydrogens, hydrogen-bonding nitrogens and directionally hydrogen-bonding oxygens among others, each with a partial charge.

Properly prepared molecule input files for AutoDock consist of pdb-like records for each atom, conforming to this AutoDock atom type set. Thus file preparation must include fixing a number of potential problems such as missing atoms, added waters, more than one molecule, chain breaks, alternate locations *etc*.

In the tutorial "Using AutoDock 4 with ADT", you prepared the ligand file using ADT, a graphical user interface. It is not reasonable to try to prepare thousands of ligand files using a graphical user interface. Tasks of this magnitude must be automated. In this exercise, we introduce **prepare_ligand4.py**, a python script in the AutoDockTools module, and show you how to use it in a Unix **foreach** loop. Details of its usage can be found in the **Appendix**.

Note:
The **prepare_ligand.py** script takes as input a pdb or mol2 filewhich is specified on the command line with the '**-l**' switch and writesa pdbqt file with charges, root, and rotatable bonds defined. The '**-d**'switch specifies the filename of a python dictionary that describesthe atomtypes and other attributes of the set of input files processed.This information will be used in the next exercise.

Procedure:

```
#!/bin/c s h
# $Id: ex02.csh,v 1.2 2005/01/31 00:48:01 lindy Exp$
#

# use the prepare_ligand4.py script to create pdbqt files
cd $VSTROOT/VirtualScreening/Ligands
foreach f (`ls *`)
    echo $f
    pythonsh ../../prepare_ligand4.py –l $f –d ../etc/ligand_dict.py
end
```

Type this: ⇨ **source $VSTROOT/scripts/ex02.csh**

Type this: ➡️ 2. Examine the results of this script:

```
\ls *.pdbqt |wc
\ls ../etc
```

tutorial

$VSTROOT

VSTutorial

Results

VirtualScreening

scripts

etc

Ligands

ligand_dict.py

ZINC*.pdbqt
ind.pdbqt
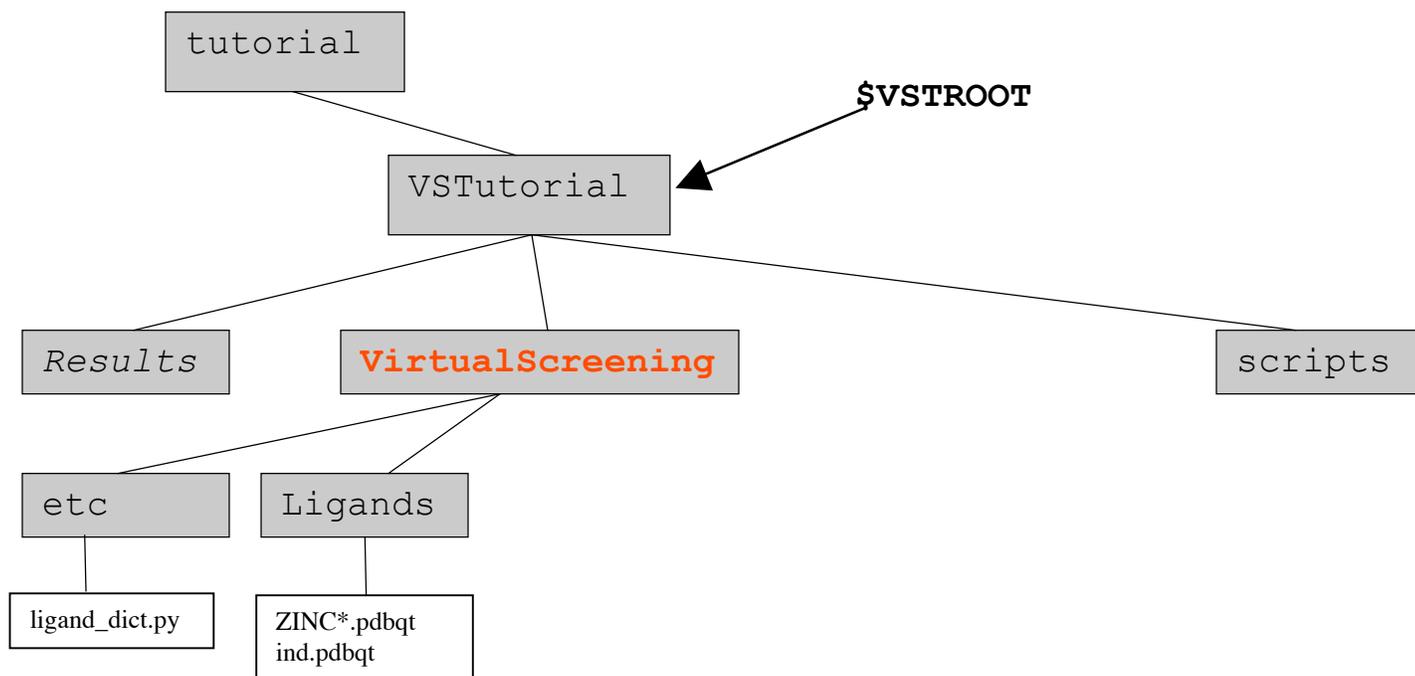
Note:
ligand_dict.py is
generated by
prepare_ligand4.py
and used in
Exercise 3.

Figure 1.5 Exercise 2 Result

3. Document:

Add an entry for this section's procedure to the **README** file. Record warning messages.

The UNIX '`script filename`' command is an alternative to the README file convention. It copies all the text from the terminal into the specified transcript file. Here, you could start a transcript before the foreach loop. To stop recording the transcript file, type **Control D.**

Exercise Three: Profiling the library: determining the covering set of Atom Types:

In docking a ligand against a receptor, AutoDock uses a special representation of the receptor: a set of grid-based potential energies files called '*grid maps*'. AutoGrid is used to pre-calculate *one* grid map for each atom type present in the ligand to be docked. A grid map consists of a three dimensional lattice of regularly spaced points, surrounding the receptor (either entirely or partly) and centered on some region of interest of the macromolecule under study.   Each point within the grid map is the sum of the pairwise potential interaction energy of a *probe* atom of a particular type with each of the atoms in the macromolecule. The 3-dimensional volume covered by the grid maps in conjunction with the 'n' active torsions in the ligand defines the 6 + 'n' dimensioned *search space*.

Note: AutoDock4 limits the number of atoms in the ligand to **2048** and the number of rotatable bonds in a ligand to **32**.

In docking a set of ligands against a single receptor, you need only *one* grid map for each atom type in the *covering set* of atom types present in the ligands. In this exercise we write a summary of the ligand library in order to determine the covering set of atom types and to exclude ligands with too many atoms, atom types, rotatable bonds, etc

Procedure:

```
#!/bin/csh
# $Id: ex03.csh,v 1.4 2005/01/31 02:23:44 lindy Exp $
# The examine_ligand_dict.py scripts reads the
# ligand_dict.pywritten in Exercise 2 and writes a summary
# describing the set ofligands  to stdout.

cd $VSTROOT/VirtualScreening/etc
cp ../../examine_ligand_dict.py .
./examine_ligand_dict.py > summary.txt
```

Notice the covering set of atoms. You may decide to remove some stems based on this information.

Type this: ➡   **source $VSTROOT/scripts/ex03.csh**

```
tutorial


                VSTutorial            $VSTROOT


  Results      VirtualScreening                    scripts


  etc      Ligands


summary.txt
```
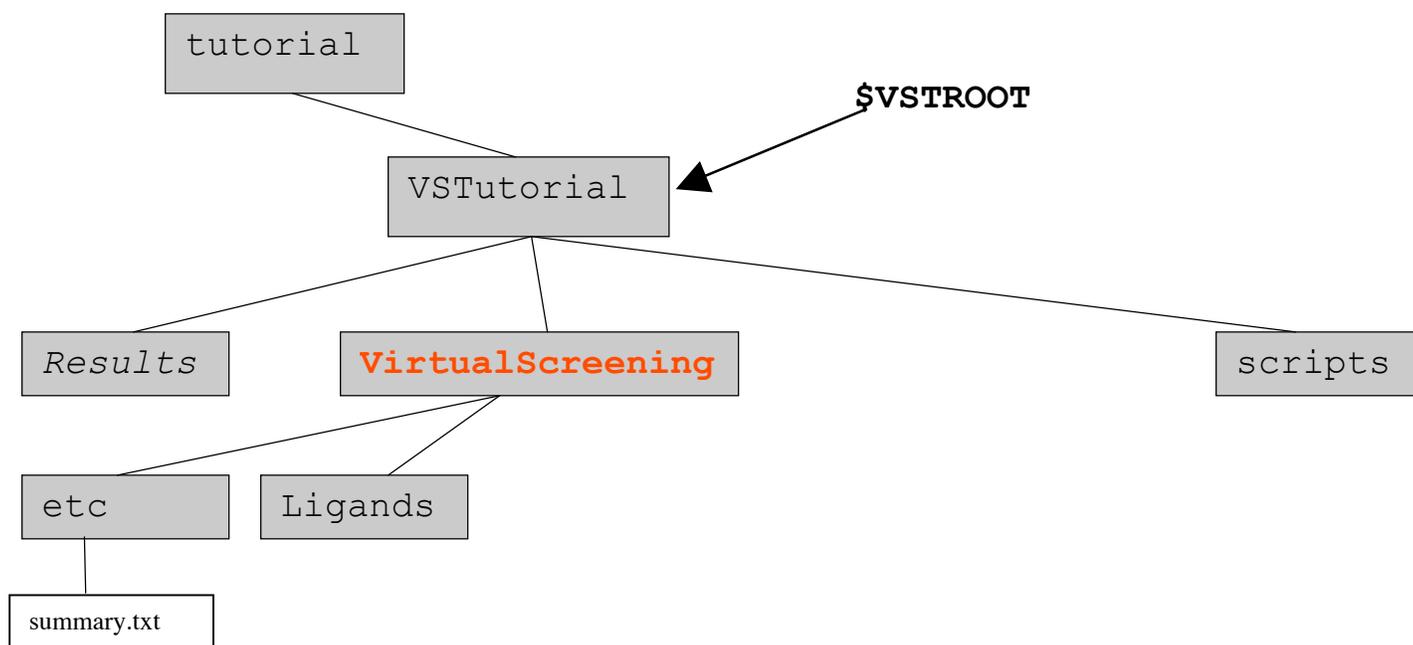
Figure 1.6 Exercise 3 Result

2. Examine the kinds of information in **summary.txt** to get an overview of the library. You should always try to have an overview of the library you are using.

Type this: ⇨   **more summary.txt**

3. Add an entry for this section's procedure to the **README** file. Note, for example, the atom types found, the range of torsion numbers….

Exercise Four: Preparing the receptor: pdb to pdbqt.

Note: For most atoms, the **autodock_type** is the same as the element. The autodock_type for aromatic carbons, which for autodock are carbons in planar cycles, is **A** to distinguish them from aliphatic carbons **C**. All oxygens are assumed to be able to accept two hydrogen bond acceptors and have the autodock_type **OA**. All hydrogens are assumed to be able to be hydrogen bond donors and have the autodock_type **HD**. Sulfur and nitrogen atoms which can accept hydrogen bonds are autodock_types **SA** and **NA** respectively and are distinguished from those which cannot which have autodock_types **S** and **N**.

The receptor file used by AutoDock must be in **pdbqt** format which is pdb plus 'q' charge and 't' autodock_type. To conform to the AutoDock atom types, polar hydrogens should be present whereas non-polar hydrogens and lone pairs should be merged, each atom should be assigned a gasteiger partial charge.

The **Receptor** directory is where we process the receptor once and only once. All the ligands will refer to this single receptor. AutoDockTools should be familiar to you from the AutoDockTools tutorial.

Procedure:

```
#!/bin/csh
# $Id: ex04.csh,v 1.2 2005/01/31 00:48:01 lindy Exp $
# Create a directory called Receptor and populate it
# with the supplied x1hpv.pdb file.On your own, use
# AutoDockTools to create the pdbqs file.

cd $VSTROOT/VirtualScreening
mkdir  Receptor
cp ../x1hpv.pdb Receptor
cd  Receptor

echo "use adt to complete this exercise"
```

Type this: **source $VSTROOT/scripts/ex04.csh**
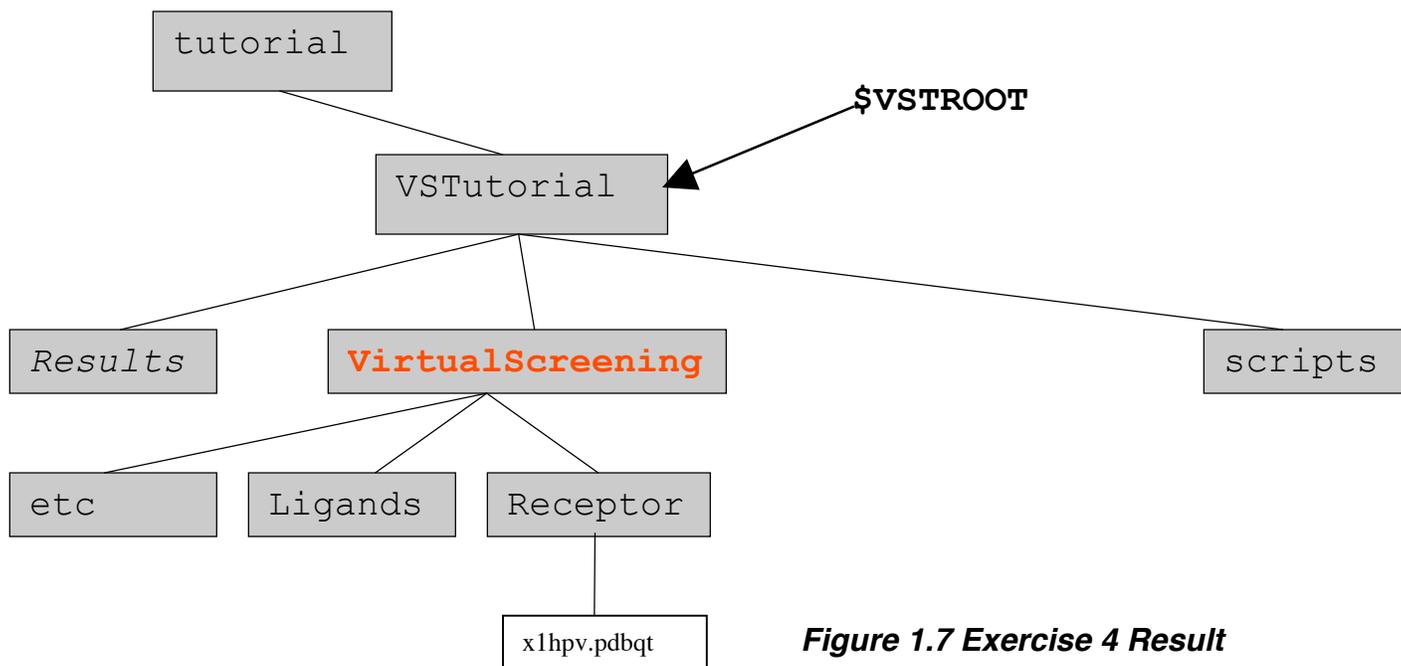
Figure 1.7 Exercise 4 Result

TSRI only start adt like this:
For linux:
/mgl/prog/share/bin/adt
For apple:
Navigate to **Applications** folder, find
AutoDockTools and double click on it.

2. Use **adt** to add hydrogens, charges and autodock types to the receptor and to write **x1hpv.pdbqt**.

adt

* **ADT -> Grid -> Macromolecule -> Open…**
- *click on – on left of* PDBQT files: (*.pdbqt)   - *button*
  to show a list other file types. Click on all files (*.)
  - *select* **x1hpv.pdb**
  - *click on* **Open**

Note: Alternatively, this preparation *could* be done via the **prepare_receptor4.py** script. However, if you are working with a single receptor, you should prepare it interactively to optimize selecting the search space.

* When processing is complete, type **x1hpv.pdbqt** into the file browser which opens. Be sure to write the file in the **Receptor** directory. Don't close **adt** because we'll use it in the next exercise.

3. Add an entry for this section's procedure to the **README** file.

Exercise Five: Preparing AutoGrid Parameter Files for the library

The grid parameter file tells **AutoGrid** the types of maps to compute, the location and extent of those maps and specifies pair-wise potential energy parameters. In general, one map is calculated for each element in the ligand plus an electrostatics map. Self-consistent 12-6 Lennard-Jones energy parameters - **Rij,** equilibrium internuclear separation and **epsij,** energy well depth - are specified for each map based on types of atoms in the macromolecule. If you want to model hydrogen bonding, this is done by specifying 12-10 instead of 12-6 parameters in the gpf.

For a library of ligands, only one atom map per ligand type is required. Each **AutoGrid4** calculation creates the set of required atom maps plus an electrostatics map and a desolvation map.

Procedure:

```
#!/bin/csh
#  $Id: ex05.csh,v 1.2 2005/01/31 00:48:01 lindy Exp $
echo "Use adt to complete this exercise (05)"
```

1. Use **adt** to write the Grid Parameter Files (gpf):
   [If you have just written the macromolecule in Exercise Four, skip this first step:
    * Grid -> Macromolecule -> Open…]

   * **Grid -> Open GPF…**
       - type in **../../x1hpv.gpf**
       - click **Open**

   * **Grid -> Set Map Types -> Directly…**
       - add these types **CL F S BR** if they are not there
       - click **Accept**

   * **Grid -> Output -> Save GPF…**
       - in the file browser, navigate down to the **Receptor** directory
       - type in **x1hpv.gpf**
       - click **Save**.

Note: we read in this reference gpf to set values for the gridcenter and the number of points.

18

2. Examine the grid parameter file you have prepared:

Type this: ➡ `cat x1hpv.gpf | more`

3. Add an entry for this section's procedure to the **README** file.

4. On Linux and Mac OS X machines, type **Control Z** followed by **bg** to put ADT in the background to continue…
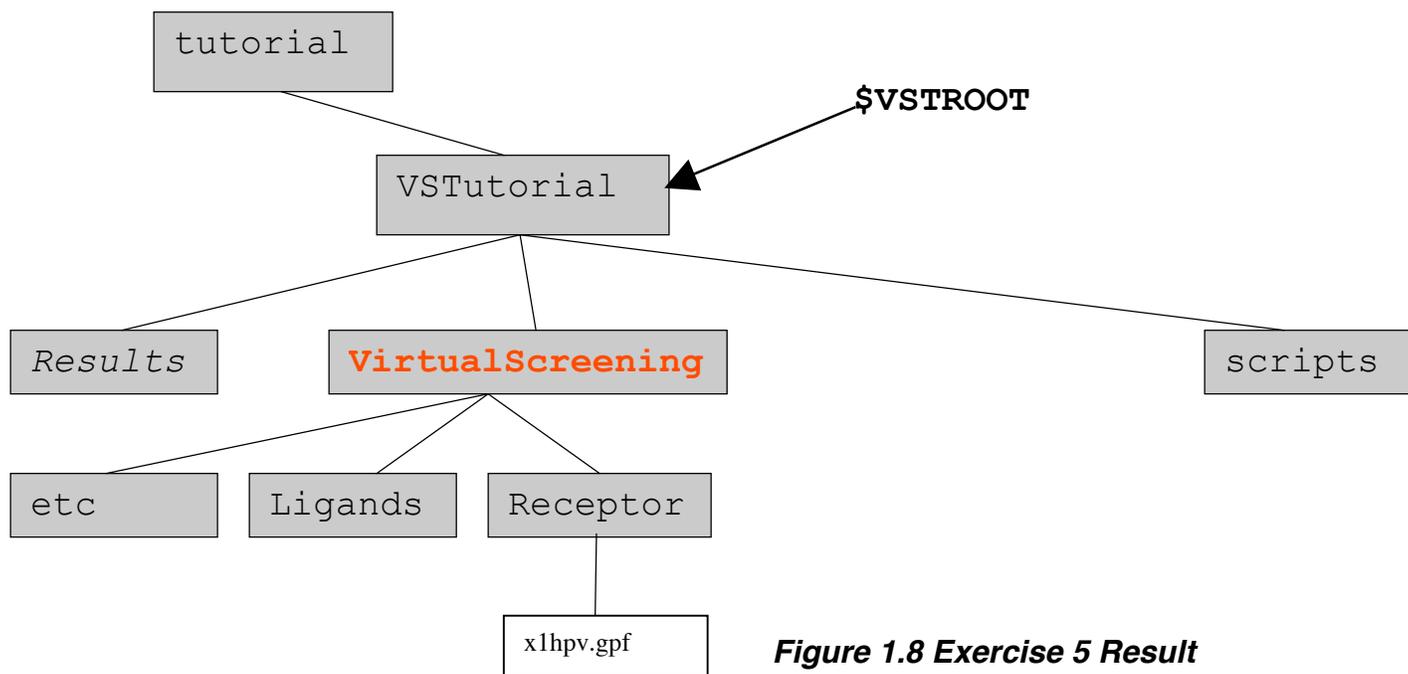
Figure 1.8 Exercise 5 Result

Exercise Six: Calculating atomic affinity maps for a ligand library using AutoGrid.

An essential part of a successful large scale computation experiment such as today's virtual screening experiment is data organization; that is, a clear directory structure should be used to organize the many input and output files. Our plan places the receptor in a separate directory which is to contain all the AutoGrid affinity maps calculated for this receptor. Under the Receptor directory will be a directory for each ligand. Each ligand directory will contain symbolic links to each map file and to the receptor.pdbqt. Each ligand directory will have its **ligand.pdbqt** file and its unique docking parameter file **x1hpv_ligand.dpf.**

In this exercise, we invoke **autogrid4** to calculate the required set of atom maps for **x1hpv**. We'll work in the Receptor directory and create the symbolic links for each ligand later.

Procedure:

```
#!/bin/csh
#$Id: ex06.csh,v 1.2 2007/05/09 00:48:01 lindy Exp $
# 1. Use autogrid4 to create the grid map files:

cd $VSTROOT/VirtualScreening/Receptor
autogrid4 -p x1hpv.gpf -l x1hpv.glg
```

Type this: ⟹ **source $VSTROOT/scripts/ex06.csh**

2. Check that the maps are there and check that there are 10 maps .

Type this: ⟹ **cd $VSTROOT/VirtualScreening/Receptor**
ls —alt *map
ls —alt *map |wc —l

3. Add an entry for this section's procedure to the **README** file.

```
tutorial

                    VSTutorial  ◄————— $VSTROOT


   Results      VirtualScreening                      scripts


   etc      Ligands    Receptor


                    x1hpv.glg
                    x1hpv.*.map
                    x1hpv.maps.xyz
                    x1hpv.maps.fld
```
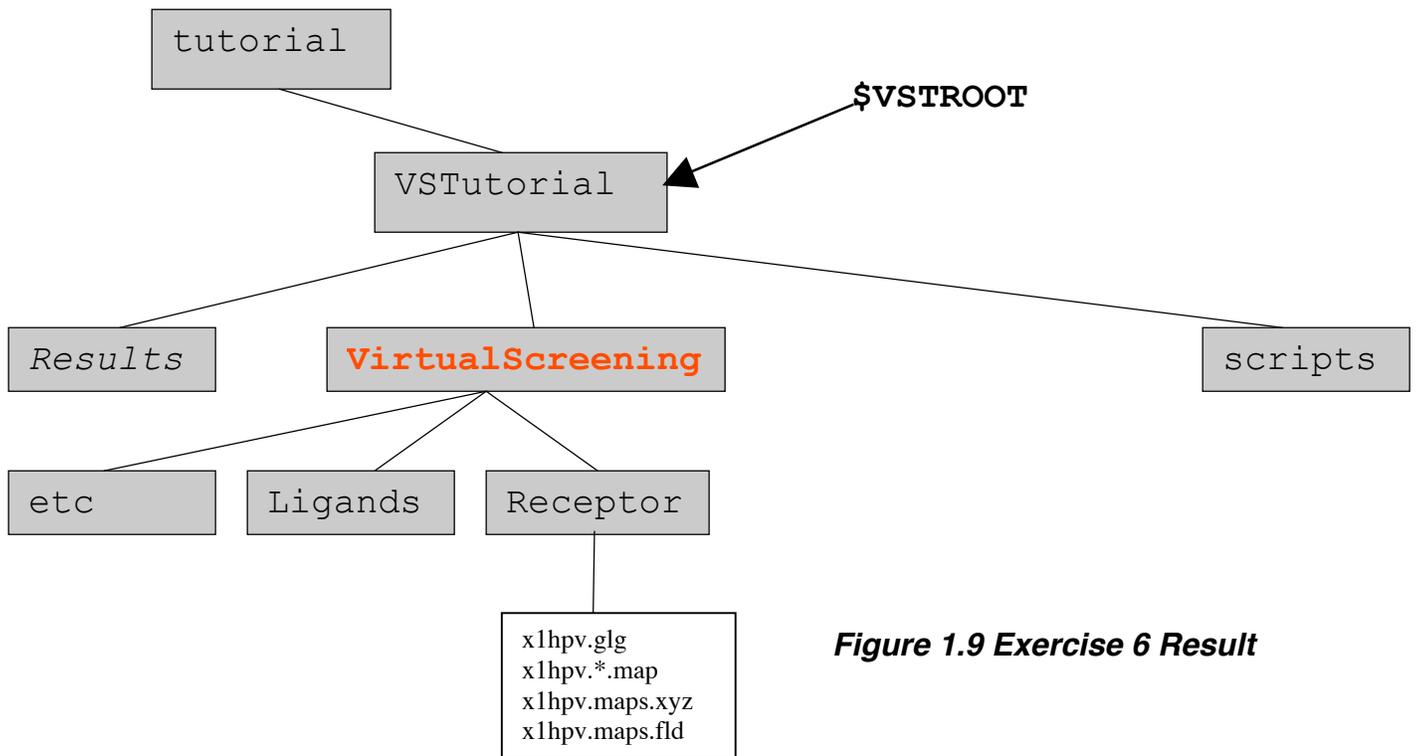
Figure 1.9 Exercise 6 Result

Exercise Seven: Validating the Protocol with a Positive Control

Before we go on and make larger and larger resource and time commitments to the virtual screening experiment, let's make sure in the next exercise that the input files are valid.

A docking "job" is a single **AutoDock** process, which carries out a number of independent docking "runs", each of which begins with the same initial conditions. The various parameters for the docking are usually stored in a docking parameter file, or "DPF". This is passed to **AutoDock** using a command line flag (-p). Each ligand requires its own docking parameter file.

To write the docking parameter files we need for today's experiment , we will use **`prepare_dpf4.py`**, a script in AutoDockTools/Utilities24. Details of its usage can be found in the Appendix.

Procedure:

```csh
#!/bin/csh
# $Id: ex07.csh,v 1.3 2005/01/31 02:23:04 lindy Exp $

# Create a directory called ind_x1hpv in the etc
#  directory:
cd $VSTROOT/VirtualScreening/etc
mkdir  ind_x1hpv
cd ind_x1hpv

# Populate the directory with the docking input files:
cp ../../Ligands/ind.pdbqt   .
ln -s ../../Receptor/x1hpv.pdbqt .
ln -s ../../Receptor/x1hpv*map* .

# Create the Docking Parameter File with modified
# parameters modified to shorten the autodock3 run time
# by restricting thesearch:
pythonsh ../../../prepare_dpf4.py -l ind.pdbqt -r x1hpv.pdbqt \
          -p ga_num_evals=25000 \
           -p ga_run=2

# Run autodock4 and examine the output:
autodock4 -p ind_x1hpv.dpf -l ind_x1hpv.dlg
```

Type this: ⟹ **source $VSTROOT/scripts/ex07.csh**

```
tutorial
```

$VSTROOT

```
VSTutorial
```

Results   **VirtualScreening**   scripts

etc   Ligands   Receptor

ind_x1hpv

x1hpv.pdbqt
x1hpv*map*
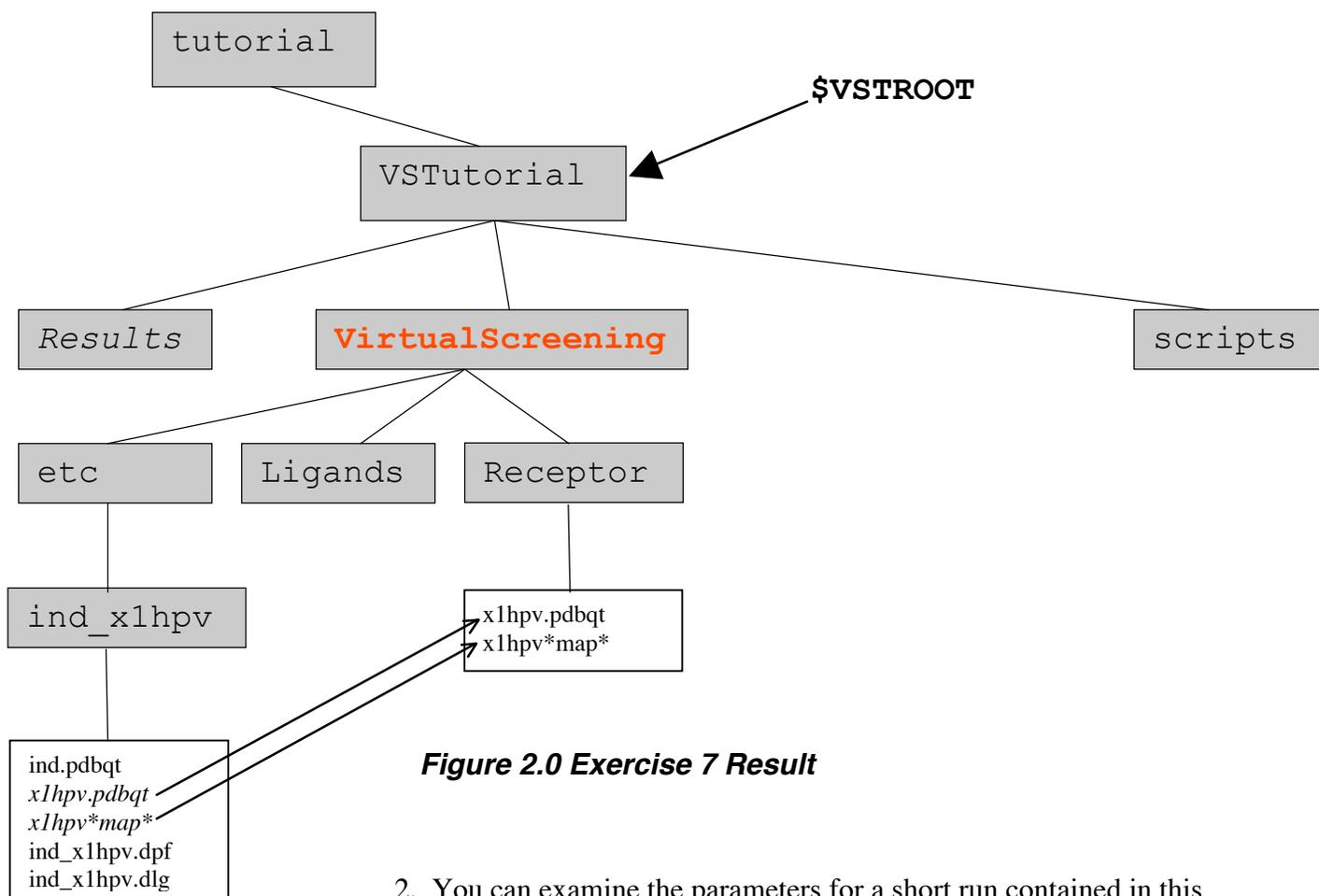
Figure 2.0 Exercise 7 Result

ind.pdbqt
x1hpv.pdbqt
*x1hpv*map**
ind_x1hpv.dpf
ind_x1hpv.dlg

2. You can examine the parameters for a short run contained in this docking parameter file.

Type this: ⟹ **cat ind_x1hpv.dpf | more**

3. Also, you can follow the execution of the autodock job using **tail.** The '-f' flag makes it follow as new output is written.

Type this: ⟹ **tail —f ind_x1hpv.dlg**

Make sure that "**Successful Completion**" is found at the end of the file.

4. Add an entry for this section's procedure to the **README** file.

Exercise Eight: Preparing the Docking Directories and Parameter Files for each ligand in a library.

In this exercise, we repeat the steps we used for the positive control in the last exercise for each ligand to be screened. There is a separate directory for each ligand. Each ligand directory contains symbolic links to the autogrid maps and to the receptor. Each ligand directory has its unique ligand.pdbqt and ligand.dpf files.

Procedure:

Suggestion: **unset echo** here if it is set because this script involves many steps for many ligands.

```
#!/bin/csh
# $Id: ex08.csh,v 1.5 2007/05/31 16:33:49 lindy Exp $
#Create the Dockings directory:

cd $VSTROOT/VirtualScreening
mkdir Dockings
cd Dockings

# Create a subdirectory named <ligand>_x1hpv and populate
# it with the docking input files: a) the pdbqt from the
# Ligands directory  will be copied directly;  b) the maps
# will be linked to the Receptor directory; and, c) the dpf
# file will be created using prepare_dpf4.py:

foreach f (`ls ../Ligands/*.pdbqt`)
set name = `basename $f .pdbqt`
echo $name
mkdir "$name"_x1hpv
cd "$name"_x1hpv
cp ../"$f" .
ln -s ../../Receptor/x1hpv.pdbqt .
ln -s ../../Receptor/x1hpv*map* .
pythonsh ../../../prepare_dpf.py -l `basename $f` -r x1hpv.pdbqt \
      -p ga_num_evals=1750000 \
      -p ga_pop_size=150 \
      -p ga_run=20 \
      -p rmstol=2.0
cd ..
end
```

Note: the docking parameters here are more realistic than those used in Exercise Seven.

Type this: ➡ **source $VSTROOT/scripts/ex08.csh**
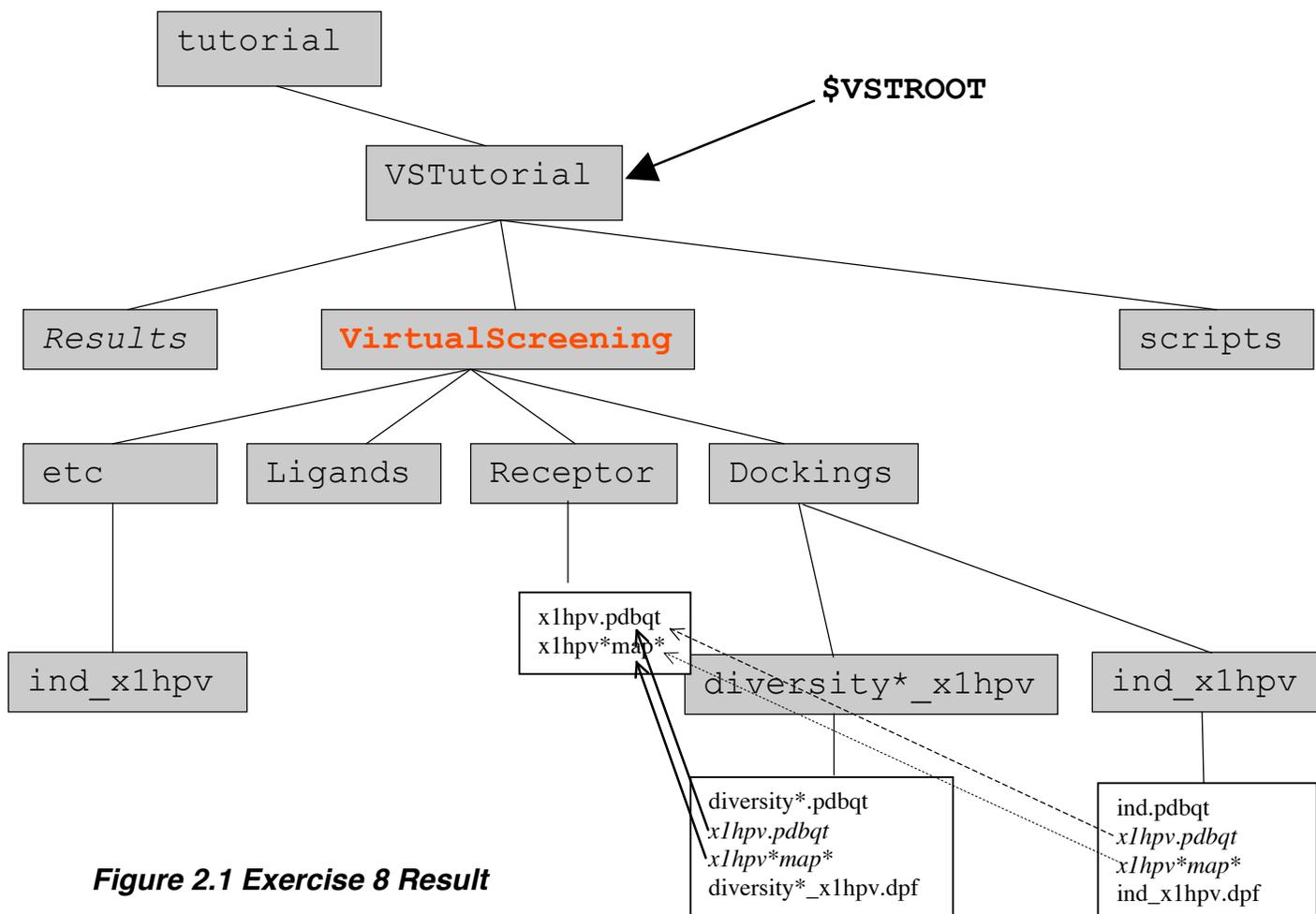
Figure 2.1 Exercise 8 Result

Note: the positive control, ind, has been processed in the same way as the ligands we are screening.

2. Examine a result of this script:

Type this: ➡ **pwd**
ls
ls | wc –l
ls ZINC00000480_x1hpv

3. Add an entry for this section's procedure to the **README** file.

Exercise Nine: Launching many AutoDock jobs.

In this exercise we will use previously computed results.

Procedure:

```
#!/bin/csh
#$Id:  ex09.csh,v 1.4 2004/12/09 02:25:23 lindy Exp $
# 1. Create a file with a list of the dockings to run:
cd $VSTROOT/VirtualScreening/Dockings
/bin/ls > ../etc/docking.list

# 2. For the purposes of this tutorial, instead of running
#autodock3 as you normally would, simply copy the results
# of a docking that we've done for you previously.

foreach d (`cat ../etc/docking.list`)
    echo $d
    cd $d
    echo "autodock4 –p $d.dpf –l $d.dlg "
    # submit4.py $d 1
    cp ../../../Results/dlgs/"$d".dlg .
     cd ..
    end
```

Note: We are copying dlg files here, NOT creating new ones with autodock.

Type this: ⇨   **source $VSTROOT/scripts/ex09.csh**

Specific details on editing this script to launch actual computations are compute-resource dependent. Here at Scripps:

- to run locally
    replace
        echo "autodock4 –p $d.dpf –l $d.dlg
    with
        autodock4 –p $d.dpf –l $d.dlg
- to run on garibaldi (see section "Using the TSRI cluster: garibaldi" for details)
    replace
        # submit4.py $d 1
    with
        submit4.py $d 1

tutorial

$VSTROOT

VSTutorial

Results   **VirtualScreening**   scripts

etc   Ligands   Receptor   Dockings

ind_x1hpv   ZINC*_x1hpv   ind_x1hpv

ZINC*_x1hpv.dlg   ind_x1hpv.dlg
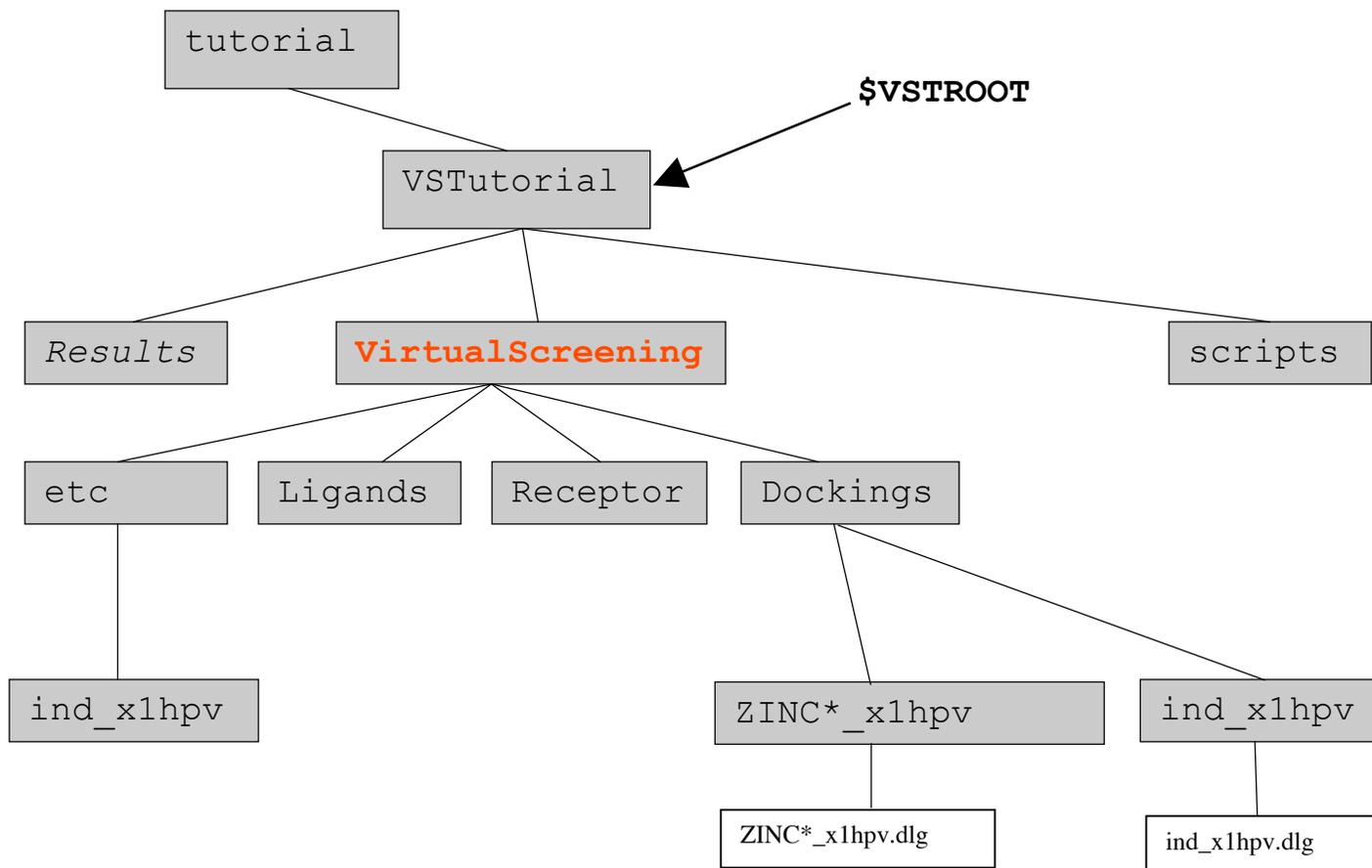
Figure 2.2 Exercise 9 Result

2. Check that the docking logs exist in the directories under the Dockings directory:

Type this:

```
cd  $VSTROOT/VirtualScreening/Dockings
ls —alt /ZINC00000480_x1hpv
```

3. Add a section for this exercise's procedure to the **README** file.

Exercise Ten: Identifying the Interesting Results to Analyze.

Note: Clustering and energy are two measures of the success of a docking. Here we record the overall lowest energy and the lowest energy conformation in the largest cluster.

The first step of analyzing the results is to build a list sorted by energy of the lowest energy docking for each ligand. To do this we first collect all the lig_rec.NNN.dlg to get **lig_rec.energies** and then sort the lig_rec.energies to create the file **lig_rec.energies.sort**

Procedure:

```
#!/bin/csh
# $Id: ex10.csh,v 1.4 2007/06/31 02:27:03 lindy Exp $#
# Extract the Free Energy of Binding for the lowest energy
# in the largest cluster from the dlg files using the python
# script summarize_results4.py:

cd $VSTROOT/VirtualScreening/Dockings
foreach d (`/bin/ls`)
  echo $d
  pythonsh ../../summarize_results4.py –d $d –t 2. –L –a –o ../etc/summary_2.0.txt
  end


# Sort the summary_2.0.txt file based on the lowest energy conformation in
the largest cluster to find your best dockings:

cd ../etc
cat summary_2.0.txt|sort –k5n  –t, > summary_2.0.sort
```

Note:
 -k5n means sort on field **5** here the lowest energy in the largest cluster.
-t, means to use commas as field separators

Type this: ➡ **source $VSTROOT/scripts/ex10.csh**
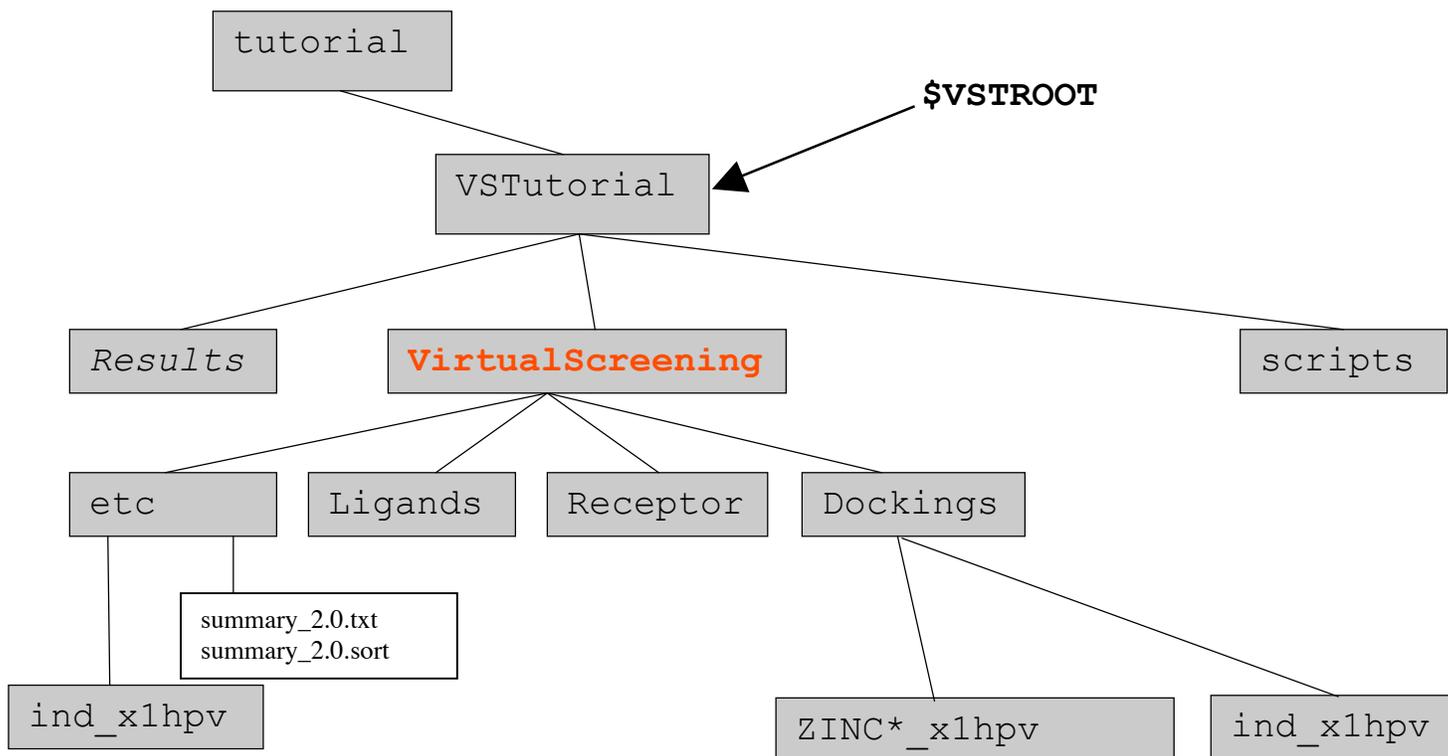
Figure 2.3 Exercise 10 Result

2. Find your ligands which bind with the lowest energy (best binders) at the top of the list in all_energies.sort. Locate the positive control. Note the ligands that have better energies than it.

```
cd ../etc
head summary_2.0.sort
```

3. Add an entry for this section's procedure to the **README** file.

Exercise Eleven: Examine Top Dockings.

1. Start adt:
cd ../Dockings
adt

2. Setup viewer:
* File->Preferences->Set Commands to be Applied on Objects
    - select **colorByAtomType** in the **Available commands** list
    - click on '**>>**'
    - click on '**OK**'

3. Setup receptor:
* Read in the file:
 File->Read Molecule
    -click on "**PDB files** (*.pdb)"
    -select "**AutoDock files (pdbqt)** (*.pdbqt)
    -select "**x1hpv.pdbqt**"

* Set center of rotation:
    -click on **pointing finger** icon
    -click on **PCOM level: Molecule** and select **Atom** from list
    -click on bar at right of "**printNodeNames**" and select **centerOnNodes**"
    -draw a box around the water residue
  (Box should be YELLOW if PCOM level is Atom).

* Display msms surface:
    Compute->Molecular Surface->Compute Molecular Surface
     -click on **OK** in **MSMS Parameters Panel** widget
    Color->by Atom Type
     -click on **MSMS-MOL** and click on **OK**

* Make molecular surface transparent using the **DejaVu GUI**:
    -click on **Sphere/Cube/Cone** (DejaVuGUI) Button
    -click on **+ root**
    -click on **+ x1hpv**
    -select **MSMS-MOL**

-click on **Material: Front**
-change **Opacity** to **.7**
-click on **Material: None**
-select **root** as current object in viewer
* Close the **DejaVu GUI**:
    -click on **Sphere/Cube/Cone** (DejaVuGUI) Button
* ADJUST the view:
    -**SHIFT-middle button** to zoom in on x1hpv's **water**

4. Repeat the following steps for each docking to be evaluated. Here we show the procedure using **ZINC00057384_x1hpv.dlg** as an example:

1. **Analyze-> Docking Logs->Open**
    -select **ZINC00057384_x1hpv.dlg**
    -click on **Open**
    -click on **OK**

2. **Analyze-> Clusterings->Show**
    write a printable version of histogram:
        -click on histogram's **Edit-> Write**
        -type in this filename: "**ZINC00057384_x1hpv.ps**"
        -click on **Save**

3. Visualize the lowest energy docked conformation
        -type '**d**' in the viewer to turn off depth-cueing
        -click on lowest energy bin in the histogram to
            open the player
        -click on right arrow to set ligand to **1_1** conformation
        Assess:
                -is the ligand in a **pocket**?
                -is each atom in the ligand in **a chemically
                    favorable position**?
        Show hydrogen bonds:
                -click on player's **ampersand** for **play options**
                -click on **Build H-bonds** and **Show Info**
                -Record number of hbonds formed
                -Display Distance(1.741) or Energy(-5.931)

3. Clea n-up for next docking log
        -click on **Close** on '**ZINC00057384**' widget
        -click on **File->Exit** on '**ZINC00057384:rms=2.0
          clustering**' histogram
        -**Analyze->Dockings->Clear** to delete this docking

5. Add entries for this section's procedure to the **README** file

Using the TSRI cluster: garibaldi

All input file preparation should be done on your local computer. The interactive head node on the **garibaldi** cluster is used to transfer the files from your computer to the cluster where the calculations will be carried out. For today's tutorial, we will demonstrate launching a sample docking and then use previously computed results.

We create a tar file of the **VSTutorial** directory tree:

Type this: ➡

```
cd /usr/tmp/tutorial
tar —czvf VSTutorial.tar.gz VSTutorial
```

Next transfer it to garibaldi using sftp.
```
sftp garibaldi
put VSTutorial.tar.gz
exit
```

Log on to garibaldi:
```
ssh garibaldi
```

Your environment on the garibaldi cluster must be set so that **autodock4** executable and the python script **submit4.py** are in your path. We will help you do this by editing the **.cshrc** file in your account on garibaldi. Make sure the directory /garibaldi/applications/people-b/autodock (which contains the autodock4 executable) is in your path.
```
set path=($path /garibaldi/people-b/applications/autodock)
```

Next uncompress the VSTutorial tree:
```
tar —xzvf VSTutorial.tar.gz
```

We will demonstrate the use of the **submit4.py** script by launching 2 jobs based on the positive control, indinavir.
```
cd VSTutorial/VirtualScreening/etc/ind_x1hpv
submit4.py ind_x1hpv 2
```

Notice that the jobs are named **######.garibaldi** and the name of the script which built the job is given. Here the scripts were named **ind_x1hpv001.j** and **ind_x1hpv002.j**. This naming convention is built into submit.py

The **pbs** command **qstat** is used for tracking job status:
```
qstat | grep yourname
```

The **pbs** command **qdel** is used for removing a job from the queue:

```
qdel ######.garibaldi
```

You will receive an email when each job finishes that includes information about whether the job finished successfully or not.

For sanity reasons, we will not be launching all the jobs. To do so you would use a foreach loop like this:

```
foreach f (`/bin/ls $VSTROOT/VirtualScreening/Ligands/*.pdbqt`)
      set name = `basename $f .pdbqt`
      echo $name
      cd $VSTROOT/VirtualScreening/Dockings/"$name"_x1hpv
      submit4.py $name 10
      end
```

Files for exercises:

Input Files:

     `x1hpv.pdb, ZINC.mol2, x1hpv.gpf`

Results Files

Ligand

     `<ligand>.pdbqt`

Macromolecule

     `x1hpv.pdbqt`

AutoGrid

     `x1hpv.gpf`
     `x1hpv.*.map, x1hpv.maps.fld, x1hpv.maps.xyz`

AutoDock

     `ind_x1hpv.dpf, ind_x1hpv.dlg,`
     `<ligand>_x1hpv.dpf, <ligand>_x1hpv.dlg`

Appendix A: Usage for AutoDockTools Scripts

The **python** scripts in AutoDockTools/Utilities24 module are customizable via input flags:

prepare_ligand4.py –l ligand_filename
      -l ligand filename (required)

Optional parameters include (defaults are in parentheses):

    -v verbose output (none)
    -**o** output pdbqt_filename (ligandname.pdbqt)
    -d dictionary filename to write summary information of per molecule atomtypes and number of active torsions (none)
    -A type(s) of repairs to make (none):
          bonds
          hydrogens
          bonds_hydrogens
    -**C** do not add charges (add gasteiger charges)
    -**p** preserve input charges on atom type, eg –p Zn
    -**U** cleanup type, what to merge (nphs_lps)
          nphs
          lps
          “ “
    -**B** types of bonds to allow to rotate (backbone)
         amide
         guanidinium
         amide_guanidinium
         “ “
    -**R** index for root
    -**F** check for and use largest non-bonded fragment (False)
    -**M** interactive (default is automatic write)
    -**I** string of bonds to inactivate composed of of zero-based atom indices eg 5_13_2_10 will inactivate atoms[5]-atoms[13] bond and atoms[2]-atoms[10] bond
    -Z inactivate all active torsions

prepare_receptor4.py –r filename
      -r   receptor_filename
  Optional parameters:"
      -v   verbose output
      -o   pdbqt_filename (receptor_name.pdbqt)
      -A   type(s) of repairs to make (" "):
          'bonds_hydrogens': build bonds and add hydrogens
          'bonds': build a single bond from each nonbonded atom
           to its closest neighbor
          hydrogens': add hydrogens
          'checkhydrogens': add hydrogens only if there are none already
          'None': do not make any repairs
       (default is 'checkhydrogens')
     -C   preserve all input charges ie do not add new charges
      (default is addition of gasteiger charges)
     -**p** preserve input charges on specific atom types, eg -p Zn -p Fe
     -U cleanup type:
         'nphs': merge charges and remove non-polar hydrogens
         'lps': merge charges and remove lone pairs
         'waters': remove water residues
         'nonstdres': remove chains composed entirely of
           residues of types other than the standard 20 amino acids
        'deleteAltB': remove XX@B atoms and rename XX@A atoms->XX
      (default is 'nphs_lps_waters_nonstdres')
    -**e** delete every nonstd residue from any chain
       'True': any residue whose name is not in this list:
          ['CYS','ILE','SER','VAL','GLN','LYS','ASN',
          'PRO','THR','PHE','ALA','HIS','GLY','ASP',
      will be deleted from any chain. NB: there are no
      nucleic acid residue names at all in the list.
      (default is False which means not to do this)
     -**M** mode (automatic)
      interactive (do not automatically write outputfile)

prepare_dpf4.py -l ligand_filename **–r** receptor_filename
      -l   ligand_filename
      -r   receptor_filename
  Optional parameters:
      -o   dpf_filename (ligand_receptor.dpf)
      -i   template dpf_filename
      -p   parameter_name=new_value
     -**k** list of parameters to write
      (default is genetic_algorithm_local_search_list)
      -v   verbose output

summarize_results4.py -d directory

    -d   directory

Optional parameters:

    -**t**   rmsd tolerance (default is 1.0)

    -**f**   rmsd reference filename

    (default is to use input ligand coordinates from docking log)

    -**b**   print best docking info only (default is print all)

    -**L**   print largest cluster info only (default is print all)

    -**B**   print best docking and largest cluster info only

    (default is print all)

    -**o**   output filename

    (default is 'summary_of_results')

    -**a**   append to output filename

    (default is to open output filename 'w')

    -**k** build hydrogen bonds

    -**r**   receptor filename

    -**u**   report unbound energy

    -**v**   verbose output