## User's Guide

# AutoDock

*Automated Docking of Flexible Ligands to Receptors*

***Version 3.0.5***

Garrett M. Morris
David S. Goodsell
Ruth Huey
William E. Hart
Scott Halliday
Rik Belew
Arthur J. Olson

### *Important*

AutoDock is distributed *free of charge* for academic and non-commercial use. There are some caveats, however. Firstly, since we do not receive funding to support the academic community of users, we cannot guarantee rapid (or even slow) response to queries on installation and use. While there is documentation, it may require at least some basic Unix abilities to install. If you need more support for the AutoDock code, a commercial version (with support) is available from Oxford Molecular (`http://www.oxmol.com`). If you can't afford support, but still need help:

(1) Ask your local system administrator or programming guru for help about compiling, using Unix/Linux, *etc.*.

(2) Consult the AutoDock web site, where you will find a wealth of information and a FAQ (Frequently Asked Questions) page with answers on AutoDock:

`http://www.scripps.edu/pub/olson-web/doc/autodock/`

(3) If you can't find the answer to your problem, send your question to the Computational Chemistry List (CCL). There are many seasoned users of computational chemistry software and some AutoDock users who may already know the answer to your question. You can find out more about the CCL on the web, at:

`http://ccl.osc.edu/ccl/welcome.html`

(4) If you have tried (1), (2) and (3), and you still cannot find an answer, send email to garrett@scripps.edu for questions about AutoGrid or AutoDock; or to rhuey@scripps.edu, for questions about AutoTors.

Thanks for your understanding!

### *E-mail addresses*

Arthur J. Olson, Ph.D.: olson@scripps.edu
Peggy Graber, *Senior Administrative Assistant*: graber@scripps.edu
Garrett M. Morris, M.A., D.Phil. garrett@scripps.edu
David S. Goodsell, Ph.D. goodsell@scripps.edu
Ruth Huey, Ph.D. rhuey@scripps.edu

### *Fax: + (858) 784-2860*



**The Scripps Research Institute,**
**Molecular Graphics Laboratory,**
**Department of Molecular Biology, Mail Drop MB-5,**
**10550 North Torrey Pines Road,**
**La Jolla, CA 92037-1000, U.S.A.**

*Modification date: November 20, 2001 2:20 pm*

# Contents

# Automated Docking



## 1. Introduction

The first version of **AutoDock**[1] was distributed to over 35 sites around the world, and that number has since grown to over 600 sites with the latest versions of AutoDock[2,3]. This user guide is the first version to accompany a significantly enhanced version of AutoDock, version 3.0, which includes powerful new search methods and a new empirical free energy function[3].

The program **AutoDock** was developed to provide an automated procedure for predicting the interaction of ligands with biomacromolecular targets. The motivation for this work arises from problems in the design of bioactive compounds, and in particular the field of computer-aided drug design. Progress in biomolecular x-ray crystallography continues to provide a number of important protein and nucleic acid structures. These structures could be targets for bioactive agents in the control of animal and plant diseases, or simply key to understanding of a fundamental aspect of biology. The precise interaction of such agents or candidate molecules is important in the development process. Indeed, **AutoDock** can be a valuable tool in the x-ray structure determination process itself: given the electron density for a ligand, **AutoDock** can help to narrow the conformational possibilities and help identify a good structure. Our goal has been to provide a computational tool to assist researchers in the determination of biomolecular complexes.

In any docking scheme two conflicting requirements must be balanced: the desire for a robust and accurate procedure, and the desire to keep the computational demands at a reasonable level. The ideal procedure would find the global minimum in the interaction energy between the substrate and the target protein, exploring all available degrees of freedom (DOF) for the system. However, it must also run on a laboratory workstation within an amount of time comparable to other computations that a structural researcher may undertake, such as a crystallographic refinement. In order to meet these demands a number of docking techniques simplify the docking procedure. Still one of the most common techniques in use today is manually-assisted docking. Here, the internal and orientational degrees of freedom in the substrate are under interactive control. While the energy evaluation for such techniques can be sophisticated, the global exploration of configurational space is limited. At the other end of the spectrum are automated methods such as exhaustive search and distance geometry. These methods can explore configurational space, but at the cost of a much simplified model for the energetic evaluation.

---

1. Goodsell, D.S. & Olson, A.J. (1990) "Automated Docking of Substrates to Proteins by Simulated Annealing", *Proteins: Str. Func. Genet.*, **8**, 195-202.
2. Morris, G. M., Goodsell, D. S., Huey, R. and Olson, A. J. (1996), "Distributed automated docking of flexible ligands to proteins: Parallel applications of AutoDock 2.4", *J. Computer-Aided Molecular Design*, **10**: 293-304.
3. Morris, G. M., Goodsell, D. S., Halliday, R.S., Huey, R., Hart, W. E., Belew, R. K. and Olson, A. J. (1998), "Automated Docking Using a Lamarckian Genetic Algorithm and and Empirical Binding Free Energy Function", *J. Computational Chemistry*, **19**: 1639-1662.

The original procedure developed for **AutoDock** used a Monte Carlo (MC) simulated annealing (SA) technique for configurational exploration with a rapid energy evaluation using grid-based molecular affinity potentials. It thus combined the advantages of exploring a large search space and a robust energy evaluation. This has proven to be a powerful approach to the problem of docking a flexible substrate into the binding site of a static protein. Input to the procedure is minimal. The researcher specifies a rectangular volume around the protein, the rotatable bonds for the substrate, and an arbitrary or random starting configuration, and the procedure produces a relatively unbiased docking.

## 2. Overview of the Method

Rapid energy evaluation is achieved by precalculating atomic affinity potentials for each atom type in the substrate molecule in the manner described by Goodford [4]. In the **AutoGrid** procedure the protein is embedded in a three-dimensional grid and a probe atom is placed at each grid point. The energy of interaction of this single atom with the protein is assigned to the grid point. An affinity grid is calculated for each type of atom in the substrate, typically carbon, oxygen, nitrogen and hydrogen, as well as a grid of electrostatic potential, either using a point charge of +1 as the probe, or using a Poisson-Boltzmann finite difference method, such as DELPHI [5,6]. The energetics of a particular substrate configuration is then found by tri-linear interpolation of affinity values of the eight grid points surrounding each of the atoms in the substrate. The electrostatic interaction is evaluated similarly, by interpolating the values of the electrostatic potential and multiplying by the charge on the atom (the electrostatic term is evaluated separately to allow finer control of the substrate atomic charges). The time to perform an energy calculation using the grids is proportional only to the number of atoms in the substrate, and is independent of the number of atoms in the protein.

The docking simulation is carried out using one of a number of possible search methods. The original AutoDock supported only one search method, although version 3.0 now has several.

The original search algorithm was the *Metropolis method*, also known as *Monte Carlo simulated annealing*. With the protein static throughout the simulation, the substrate molecule performs a random walk in the space around the protein. At each step in the simulation, a small random displacement is applied to each of the degrees of freedom of the substrate: translation of its center of gravity; orientation; and rotation around each of its flexible internal dihedral angles. This displacement results in a new configuration, whose energy is evaluated using the grid interpolation procedure described above. This new energy is compared to the energy of the preceding step. If the new energy is lower, the new configuration is immediately accepted. If the new energy is higher, then the configuration is accepted or rejected based upon a probability expression dependent on a user defined temperature, *T*. The probability of acceptance is given by:

4. Goodford, P.J. (1985) “A Computational Procedure for Determining Energetically Favorable Binding Sites on Biologically Important Macromolecules”, *J. Med. Chem.*, **28**, 849-857.
5. Sharp, K., Fine, R. & Honig, B. (1987) *Science*, **236**, 1460-1463.
6. Allison, S.A., Bacquet, R.J., & McCammon, J. (1988) *Biopolymers*, **27**, 251-269.

$$P(\Delta E) = e^{\left(-\frac{\Delta E}{k_B T}\right)}$$

where $\Delta E$ is the difference in energy from the previous step, and $k_B$ is the Boltzmann constant. At high enough temperatures, almost all steps are accepted. At lower temperatures, fewer high energy structures are accepted.

The simulation proceeds as a series of cycles, each at a specified temperature. Each cycle contains a large number of individual steps, accepting or rejecting the steps based upon the current temperature. After a specified number of acceptances or rejections, the next cycle begins with a temperature lowered by a specified schedule such as:

$$T_i = gT_{i-1}$$

where $T_i$ is the temperature at cycle $i$, and $g$ is a constant between 0 and 1.

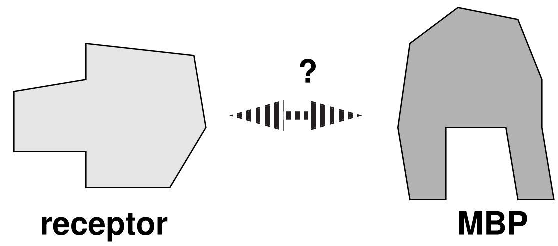
Simulated annealing allows an efficient exploration of the complex configurational space with multiple minima that is typical of a docking problem. The separation of the calculation of the molecular affinity grids from the docking simulation provides a modularity to the procedure, allowing the exploration of a range of representations of molecular interactions, from constant dielectrics to finite difference methods and from standard 12-6 potential functions to distributions based on observed binding sites.

# 3. Applications

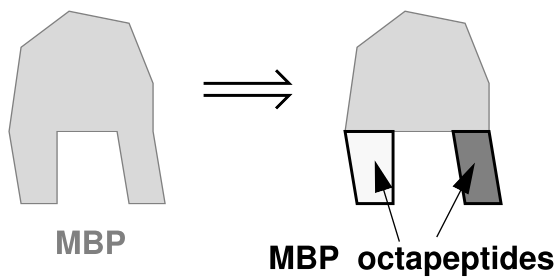
The original FORTRAN version of **AutoDock** was initially tested on a number of protein-substrate complexes which had been characterized by x-ray crystallography[7]. These tests included phosphocholine binding in a antibody combining site, N-formyltryptophan binding to chymotrypsin and N-acetylglucosamine binding to Lysozyme. In almost all cases the results of the **AutoDock** simulations functionally reproduced the crystallographic complexes. In further applications **AutoDock** was used to predict interactions of substrates with aconitase prior to any crystallographic structures for complexes. In this work we not only predicted the binding mode of isocitrate, but we demonstrated the utility of **AutoDock** in generating substrate models during the early stages of crystallographic proteins structure refinement [8]. Citrate docking experiments showed two binding modes, one of which approximated the experimental electron density determined for an aconitase-nitrocitrate complex. The docking simulation results provided insight into the proposed reaction mechanism of the enzyme.

7. Goodsell, D.S. & Olson, A.J. (1990) “Automated Docking of Substrates to Proteins by Simulated Annealing”, *Proteins: Str. Func. Genet.*, **8**, 195-202.
8. Goodsell, D.S., Lauble, H., Stout, C.D & Olson, A.J. (1993) “Automated Docking in Crystallography: Analysis of the Substrates of Aconitase”, *Proteins: Str. Func. Genet.,* **17**, 1-10.

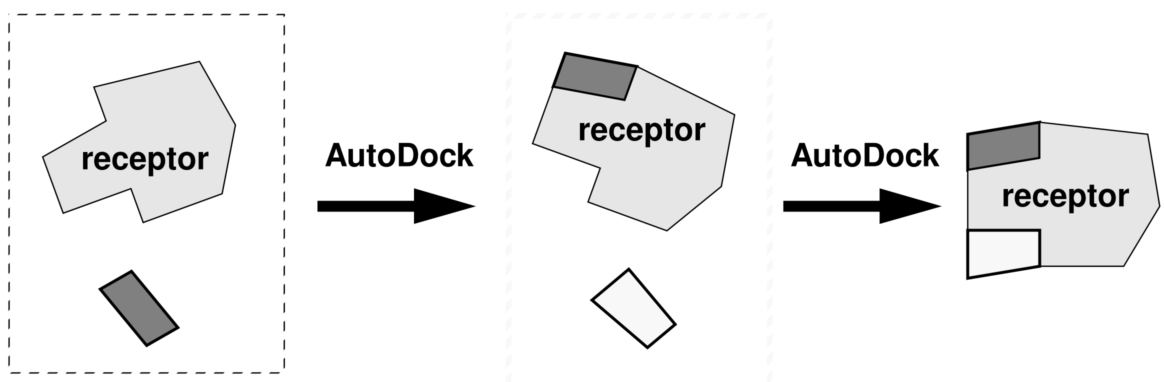
One novel and intriguing use of the software was reported from Koshland's laboratory[9]. These



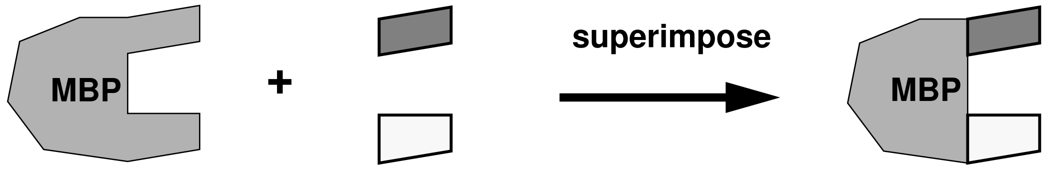

investigators used the known structures of the maltose-binding protein (MBP) and the ligand binding domain of the aspartate receptor to predict the structure of the receptor-protein complex (see diagram below). They used knowledge from mutational studies on MBP to select two




octapeptides on the protein known to be involved in the binding to the aspartate receptor, which

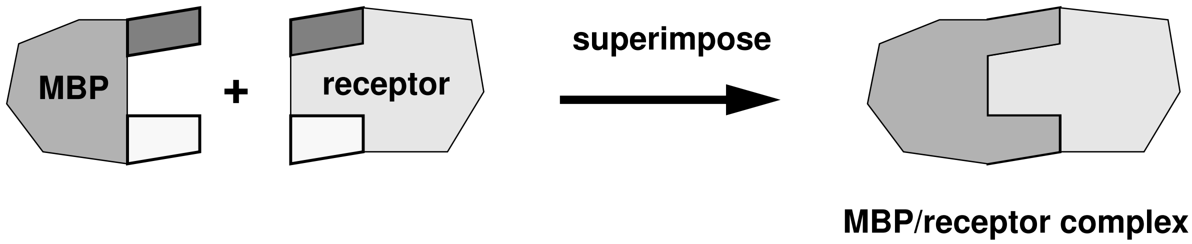



they docked independently to the model of the receptor using our automated docking code (the backbones of the peptides were fixed, but the side-chain conformations and overall orientations were unrestrained).




9. Stoddard, B.L. & Koshland, D.E. (1992) "Prediction of a receptor protein complex using a binary docking method.", *Nature*, **358** (6389), 774-776.

The distance and orientation of the two peptides as docked to the receptor corresponded to that in the intact MBP, thus enabling a reasonable prediction of the protein-receptor complex. This technique could be generally useful in situations where there are data on multi-site interactions.




# 4. What's New

In **AutoDock** version 3.0, we have added a promising new, hybrid search technique that implements an *adaptive* global optimizer with local search, based on the work of **Rik Belew** and **William Hart**[10] at the Department of Computational Science, University of California at San Diego. The global search method is a C++ implementation of a modified **genetic algorithm** (**GA**), with 2-point crossover and random mutation. The local search method is based on the optimization algorithm of **Solis and Wets**[11] (**SW**), which has the advantage that it does not require gradient information in order to proceed. The local searcher modifies the phenotype, which is allowed to update the genotype: clearly this contravenes Mendelian genetics observed in nature, but it does improve the overall performance of the method. We refer to this hybrid genetic algorithm with phenotypic local search as a **Lamarckian Genetic Algorithm** or **LGA** for short, since it utilizes (discredited) Lamarckian notion that an adaptations of an individual to its environment can be inherited by its offspring.

The SW local searcher uses fixed variances which are initially and uniformly 1. These variances are used for probabilistically determining the change to a particular state variable, like the *x*-translation. These variances are either doubled or halved during the search, depending on the number of consecutive successful or failed moves. Success is a drop in energy. We have modified the classical SW method to take into account in the variances the relative magnitudes of translations and rotations (in Angstroms and radians respectively). We call this method pseudo-Solis and Wets (**pSW**).

The genome consists of floating point genes each of which encodes one state variable describing the molecular position, orientation and conformation. This is a departure from the classical GA approach, which dictates a purely binary implementation. By setting the rate of genetic crossover to zero, and increasing the rate of genetic mutation, our hybrid GA can mimic an **evolutionary**

10. William Hart's doctoral thesis describes this hybrid global-local method, and can be found on the World Wide Wed at "http://www.cs.sandia.gov/~wehart/abstracts_html/thesis.html".
11. F.J. Solis and R.J.-B. Wets. (1981) "Minimization by random search techniques", *Mathematical Operations Research*, **6**, 19-30.

**programming** (**EP**) method.

**AutoDock** can now be used as a standalone energy minimizer by using the command "`do_local_only`". Thus the user can now minimize a structure using exactly the same force field as is used in the dockings. This could be useful, for example, in minimizing a crystal structure to relieve bad contacts.

The hybrid global-local search routine uses a library of portable routines for random number generation, based on the method of L'Ecuyer & Cote[12]. This is code is a transliteration of the original Pascal carried out by the Department of Biomathematics, University of Texas. This random number generator (RNG) has the advantage of providing a set of random numbers that are hardware independent. The simulated annealing (SA) algorithm still uses the built-in "drand48" function on most platforms, but the drand48 implementation may vary from platform to platform.

Our results show that the Lamarckian GA, (also known as the hybrid GA-SW and GA-pSW methods) reproduce the crystal complex more reliably using the same number of energy evaluations than SA does.

There are new keywords that have been added to **AutoDock** to assist in setting up a docking using the new methods. Those keywords that pertain to the genetic algorithm are prefixed with the letters "`ga_`" , those specific to local search have the prefix "`ls_`" and those specific to Solis and Wets and pseudo-Solis and Wets have the prefix "`sw_`". To use the GA, the "`set_ga`" directive must be given Classical Solis and Wets needs "`set_sw1`", and pseudo-Solis and Wets requires "`set_psw1`". In order to begin the GA, the keyword "`ga_run`" must be given along with a number of runs to be executed.

In order to perform conformational cluster analysis after the dockings, the keyword "analysis" must be supplied as the final line, otherwise no structural output will be generated.

A small change must now be made to old SA docking parameter files, with the addition of the keywords "`simanneal`" and "`analysis`". These instruct **AutoDock** respectively to begin the SA docking, and when the requested number of runs have been carried out, perform cluster analysis.

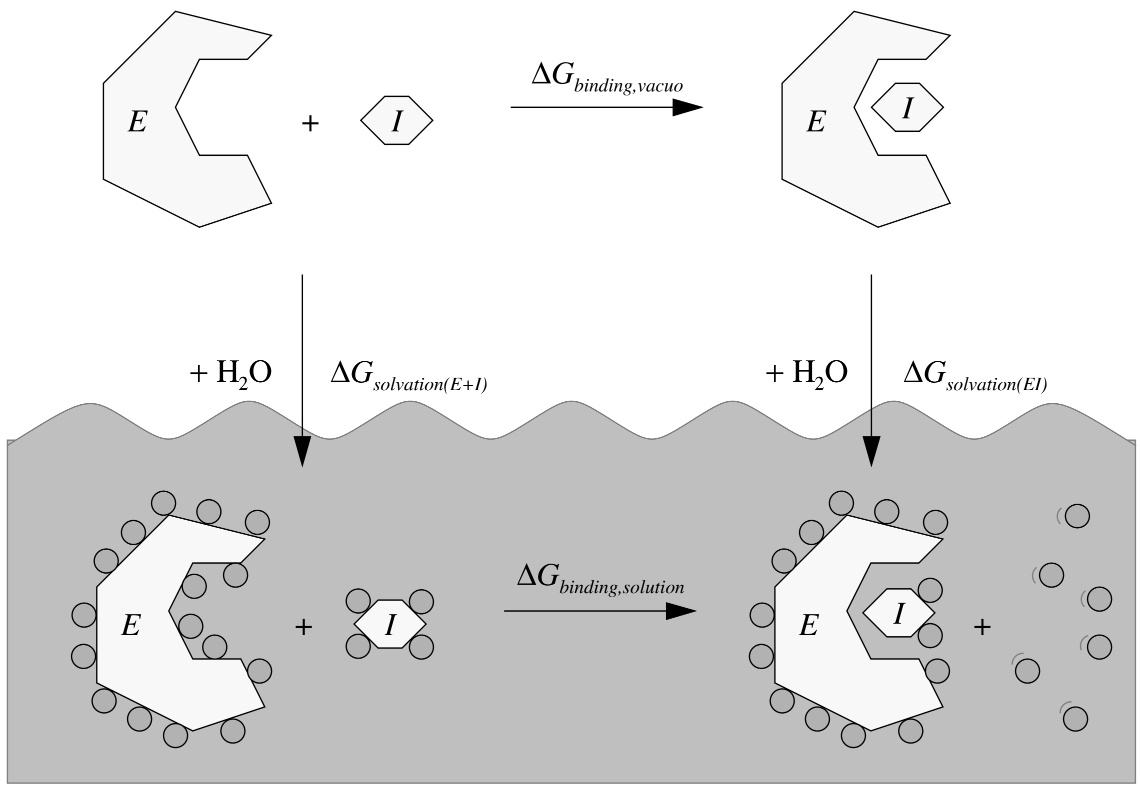
12. P. L'Ecuyer and S. Cote. (1991) "Implementing a Random Number Package with Splitting Facilities", *ACM Transactions on Mathematical Software,* **17**, 98-111

# Theory

## 5. Overview of the Free Energy Function

In version 3.0 of **AutoGrid** and **AutoDock**, we introduced a new kind of scoring function that is used during and at the end of the dockings. It is based on the principles of QSAR (quantitative structure-activity relationships) and was parameterized using a large number of protein-inhibitor complexes for which both their structure and inhibition constants, or $K_i$, were known. The user is encouraged to refer to the description of how this free energy function was derived in the original literature[3].






The above diagram shows the thermodynamic cycle for the binding of an enzyme, *E*, and an inhibitor, *I*, in both the solvated phase and *in vacuo*. Note the solvent molecules are indicated by filled circles: they tend to be ordered around the larger molecules, but when *E* and *I* bind, several solvent molecules are liberated and become disordered. This is an entropic effect and is the basis of the hydrophobic effect. The solvent ordering around *E* and *I*, when both bound and unbound, is strongly influenced by the hydrogen bonding between these molecules. These hydrogen bonds between solvent and *E,* and solvent and *I,* contribute enthalpic stabilization, and is something we can estimate in our new free energy function.

According to Hess's law of heat summation, the change in free energy between two states will be the same, no matter what the path. So we can calculate the free energy of binding in solvent by the following equation:

$$\Delta G_{binding,solution} = \Delta G_{binding,vacuo} + \Delta G_{solvation(EI)} - \Delta G_{solvation(E+I)}$$

Since we can calculate $\Delta G_{binding,vacuo}$ from our docking simulation, and can estimate the free energy change upon solvation for the separate molecules *E* and *I*, and for the complex, *EI*, $\Delta G_{solvation(EI)}$ and $\Delta G_{solvation(E+I)}$ respectively, then it is also possible to calculate the free energy change upon binding of the inhibitor to the enzyme in solution, $\Delta G_{binding,solv}$. Thus, we can estimate the inhibition constant, $K_i$, for the inhibitor, *I*.

A key point to bear in mind is that most parts of the new scoring function are essentially the same as the original AutoDock scoring function used in versions prior to 3.0, except that various terms in the molecular mechanics energy function have been re-scaled by new coefficients, and *new* terms have been introduced. These new terms include the desolvation free energy of the ligand, and an estimate of the loss of conformational degrees of freedom of the ligand upon binding.

The coefficients were derived using linear regression analysis, and we chose the linear regression model that most closely fit the observed inhibition constant data. Thus the user should not modify these coefficients lightly.

For the curious, these coefficients are defined in "`gpf3gen.awk`" and "`dpf3gen.awk`", and their variable names and values are as follows:

```
#
# Free energy model 140n coefficients:
#
FE_vdW_coeff   = 0.1485
FE_estat_coeff = 0.1146
FE_hbond_coeff = 0.0656
FE_tors_coeff  = 0.3113
FE_desol_coeff = 0.1711
```

So for example, in Table 2 on page 19 and Table 3 on page 21, the values used in the **AutoDock** 3.0 scoring function will be as given except the van der Waals coefficients and well depth energies, $\varepsilon$, will be scaled by 0.1485, the electrostatic energy will be scaled by 0.1146, and the hydrogen bonding terms will be scaled by 0.0656. The new terms for loss of torsional degrees of freedom upon binding and the ligand desolvation free energy will be scaled by 0.3113 and 0.1711 respectively. The torsional term is actually the number of rotatable bonds in the ligand that rotate heavy atoms, multiplied by the coefficient, 0.3113. Hydroxyl rotors, for example, are not counted. This number is actually determined by AutoTors, and is written in the ligand PDBQ file after the new keyword, "`TDOF`". This can also be set in the DPF, using the keyword "`torsdof` *n* `0.3113`", where *n* is the number of heavy-atom rotatable bonds.

Note that the new desolvation free energy term is only calculated for aliphatic and aromatic carbon atoms in the ligand. We found that the quality of the final empirical free energy model was not affected by the inclusion of the heteroatoms N and O in this term, so these are ignored for simplicity and speed of calculation. This does mean that the ligand input PDBQ file must distinguish between carbon atoms that are aliphatic and those that are aromatic. This is done by changing the atom names of the aromatic carbons so that the initial 'C' is replaced by 'A'. For example, if the ligand happened to be a peptidomimetic inhibitor which contained a Phe-sidechain, then the atom names in the PDBQ file would have to be changed from CG, CD1, CD2, CE1, CE2 and CZ, into AG, AD1, AD2, AE1, AE2 and AZ, respectively. To help the user do this automatically, **AutoTors** has a new option that can be invoked using the '-A' flag on the command line. This will look for all planar cyclic carbons and assume these are aromatic: it will then change their atom names automatically. The user can also override the default angle **AutoTors** uses to determine planarity. See the Section below on **AutoTors**.

This also means **AutoGrid** will calculate two different types of carbon grid maps, one for aliphatic carbons in the ligand (*.C.map), and one for aromatic carbons in the ligand (*.A.map). These will be input by **AutoDock** for the docking calculations, and the internal energy parameters must also specify the values for the aromatic and aliphatic carbon atoms.

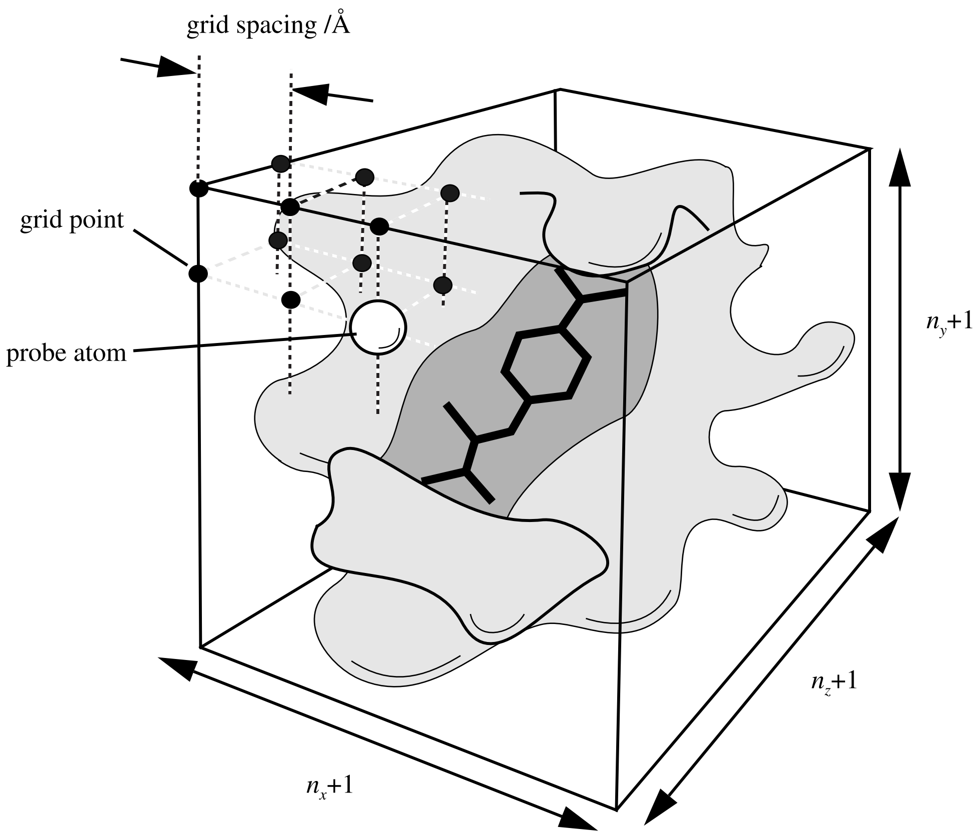
Another point to bear in mind is that **AutoGrid** 3.0 can calculate smoothed pairwise potentials: the lowest energy within a user-defined distance is stored at the current position. This has the effect of 'widening' the basin of affinity. This smoothing distance is set using the keyword '`smooth`' in the grid parameter input file to **AutoGrid**. It is very important that the value of '`smooth`' is not changed from 0.5Å, since the free energy function was calibrated using this setting. If smooth is set to 0.0Å, for example, the calculated free energies will probably be too high.

# 6. Grid Maps

**AutoDock** requires pre-calculated *grid maps*, one for each atom type present in the ligand being docked. This helps to make the docking calculations extremely fast. These maps are calculated by **AutoGrid**. A grid map consists of a three dimensional lattice of regularly spaced points, surrounding (either entirely or partly) and centered on some region of interest of the macromolecule under study. This could be a protein, enzyme, antibody, DNA, RNA or even a polymer or ionic crystal. Typical grid point spacing varies from 0.2Å to 1.0Å, although the default is 0.375Å (roughly a quarter of the length of a carbon-carbon single bond). Each point within the grid map stores the potential energy of a '*probe*' atom or functional group that is due to all the atoms in the macromolecule.

The user must specify an *even* number of grid points in each dimension, $n_x$, $n_y$ and $n_z$. This is because **AutoGrid** adds a central point, and **AutoDock** requires an odd number of grid points. The probe's energy at each grid point is determined by the set of parameters supplied for that particular atom type, and is the summation over all atoms of the macromolecule, within a non-bonded cutoff radius, of all pairwise interactions.

The following figure illustrates the main features of a grid map:




The ligand can be seen in the centre of the grid map, buried inside the active site of the protein. In this case, the grid map encompasses the whole protein. The grid spacing is the same in all three dimensions.

As mentioned in the description of the new free energy function, the user can smooth the pairwise potentials, by storing the lowest energy within a given distance of the current pairwise separation. The value of this specified in the GPF, and should not be changed from '`smooth 0.5`' in order to use the function described in the literature[3].

In addition, in **AutoGrid** 3.0, the user must use a new utility program to specify the atomic fragmental volume and atomic solvation parameters for each atom in the macromolecule, and for all the carbon atoms in the ligand. This requires the assignment of these parameters to the macromolecule using the program 'addsol' to create a PDBQS file. This resembles a PDB formatted file, but in addition gives the partial charges and solvation parameters for each atom. The solvation parameters for the ligand 'probe' atoms are specified in the GPF, by the "`sol_par`" keyword, and should not be modified unless the free energy function is not needed.

One last addition to the GPF, is the introduction of a '`constant`' keyword. This was introduced to penalize hydrogen bonds lost upon ligand binding. This defines a constant energy that is added to all the values in a grid map. The rationale for this is as follows: when a ligand goes from the unbound to the bound state, and is capable of making hydrogen bonds, it may or may not lose the enthalpic stabilization of one or more of these H-bonds. We assume that a ligand in the aqueous phase accepts and donates as many hydrogen bonds as it can. However, we found that a plot of the total hydrogen bonding energy for the bound ligand in the protein complex, versus the maximum number of *possible* hydrogen bonds the ligand could form, indicated that on average only 36% of the maximum well depth stabilization was achieved for each *possible* hydrogen bond. Thus a ligand atom in the complex that has a hydrogen bonding capacity must experience at least this amount of stabilization before it can be formed. Those that cannot are penalized by this amount.

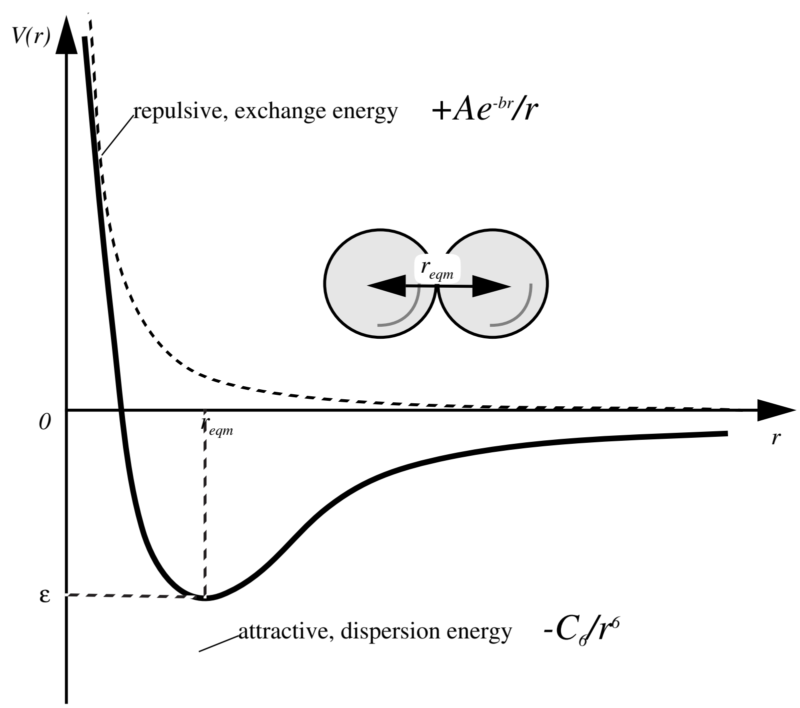
## 7. Van der Waals Potential Energy



The pairwise potential energy, *V(r)*, between two non-bonded atoms can be expressed as a function of internuclear separation, *r*, as follows,

$$V(r) = \frac{Ae^{-br}}{r} - \frac{C_6}{r^6}$$

Graphically, if $r_{eqm}$ is the *equilibrium internuclear separation*, and ε is the *well depth* at $r_{eqm}$, then:




The exponential, repulsive, exchange energy is often approximated thus,

$$\frac{A}{r}e^{-br} \approx \frac{C_{12}}{r^{12}}$$

Hence pairwise-atomic interaction energies can be approximated using the following general equation,

$$V(r) \approx \frac{C_n}{r^n} - \frac{C_m}{r^m} = C_n r^{-n} - C_m r^{-m}$$

Theory

where $m$ and $n$ are integers, and $C_n$ and $C_m$ are constants whose values depend on the depth of the energy well and the equilibrium separation of the two atoms' nuclei. Typically the 12-6 Lennard-Jones parameters ($n$=12, $m$=6) are used to model the Van der Waals' forces[13] experienced between two instantaneous dipoles. However, the 12-10 form of this expression ($n$=12, $m$=10) can be used to model hydrogen bonds (see *"Modeling Hydrogen Bonds"* below). Appendix II gives the parameters which were distributed with the first (FORTRAN-77) version of **AutoDock**, and which have been used in numerous published articles.

A revised set of parameters has been calculated, which use the same Van der Waals radius of a given atom for all pairwise distances, no matter what the other atom. Likewise, the well-depths are consistently related. Let $r_{eqm,\ XX}$ be the equilibrium separation between the nuclei of two like atoms, $X$, and let $\varepsilon_{XX}$ be their pairwise potential energy or well depth. The combining rules for the Van der Waals radius, $r_{eqm}$, and the well depth, $\varepsilon$, for two different atoms $X$ and $Y$, are:

$$r_{eqm,XY} = \tfrac{1}{2}(r_{eqm,XX} + r_{eqm,YY})$$

$$\varepsilon_{XY} = \sqrt{\varepsilon_{XX}\varepsilon_{YY}}$$

A derivation for the Lennard-Jones potential sometimes seen in text books invokes the parameter, $\sigma$, thus,

$$r_{eqm,\ XY} = 2^{\frac{1}{6}}\sigma$$

Then the Lennard-Jones 12-6 potential becomes:

$$V_{12\text{-}6}(r) = 4\varepsilon_{XY}\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

Hence, the coefficients $C_{12}$ and $C_6$ are given by:

13. van der Waals, J. H. (1908) Lehrbuch der Thermodynamik, Mass and Van Suchtelen, Leipzig, Part 1

$$C_{12} = \varepsilon_{XY} r_{eqm,\,XY}^{12}$$

$$C_6 = 2\varepsilon_{XY} r_{eqm,\,XY}^{6}$$

We can derive a general relationship between the coefficients, equilibrium separation and well depth as follows. At the equilibrium separation, $r_{eqm}$, the potential energy is a minimumand equal to the well depth: in other words, $V(r_{eqm}) = -\varepsilon$. The derivative of the potential with respect to separation will be zero at the minimum potential:

$$\frac{dV}{dr} = -\frac{nC_n}{r^{n+1}} + \frac{mC_m}{r^{m+1}} = 0$$

therefore:

$$\frac{nC_n}{r^{n+1}} = \frac{mC_m}{r^{m+1}}$$

so:

$$C_m = \frac{nC_n r^{m+1}}{mr^{n+1}} = \frac{n}{m} C_n r^{(m-n)}$$

Substituting $C_m$ into the original equation for $V(r)$, then at equilibrium we obtain,

$$-\varepsilon = \frac{C_n}{r_{eqm}^n} - \frac{nC_n r_{eqm}^{(m-n)}}{mr_{eqm}^m}$$

Rearranging:

$$C_n\left(\frac{mr_{eqm}^m - nr_{eqm}^n r_{eqm}^{(m-n)}}{mr_{eqm}^n r_{eqm}^m}\right) = -\varepsilon$$

Therefore, the coefficient $C_n$ can be expressed in terms of $n$, $m$, $\varepsilon$ and $r_{eqm}$ thus:

$$\boxed{C_n = \frac{m}{n-m}\varepsilon r_{eqm}^n}$$

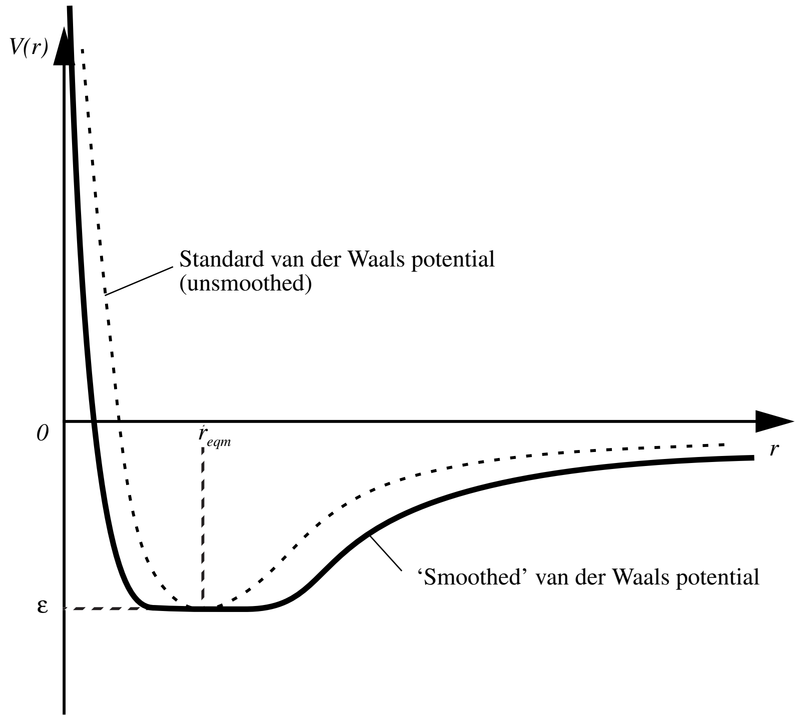
and, substituting into original equation for $V(r)$,

$$\boxed{C_m = \frac{n}{n-m}\varepsilon r_{eqm}^m}$$

In summary, then, we obtain the general equation for any *n*, *m*:

$$V(r) \approx \frac{\frac{m}{n-m}\varepsilon r_{eqm}^{n}}{r^{n}} - \frac{\frac{n}{n-m}\varepsilon r_{eqm}^{m}}{r^{m}}$$

One final point worth making here is the effect of the '`smooth 0.500`' command on the pairwise potentials. This is set in the **AutoGrid** input file (also known as the 'GPF'). This is best illustrated with a diagram; note that this has the effect of widening the region of maximum affinity at ε, and also reduces the potential energy at *r=0* to a finite value:




Example $r_{eqm}$ and ε parameters for various AMBER atom types of carbon are shown in Table 1.

**Table 1: AMBER parameters for carbon atom types.**

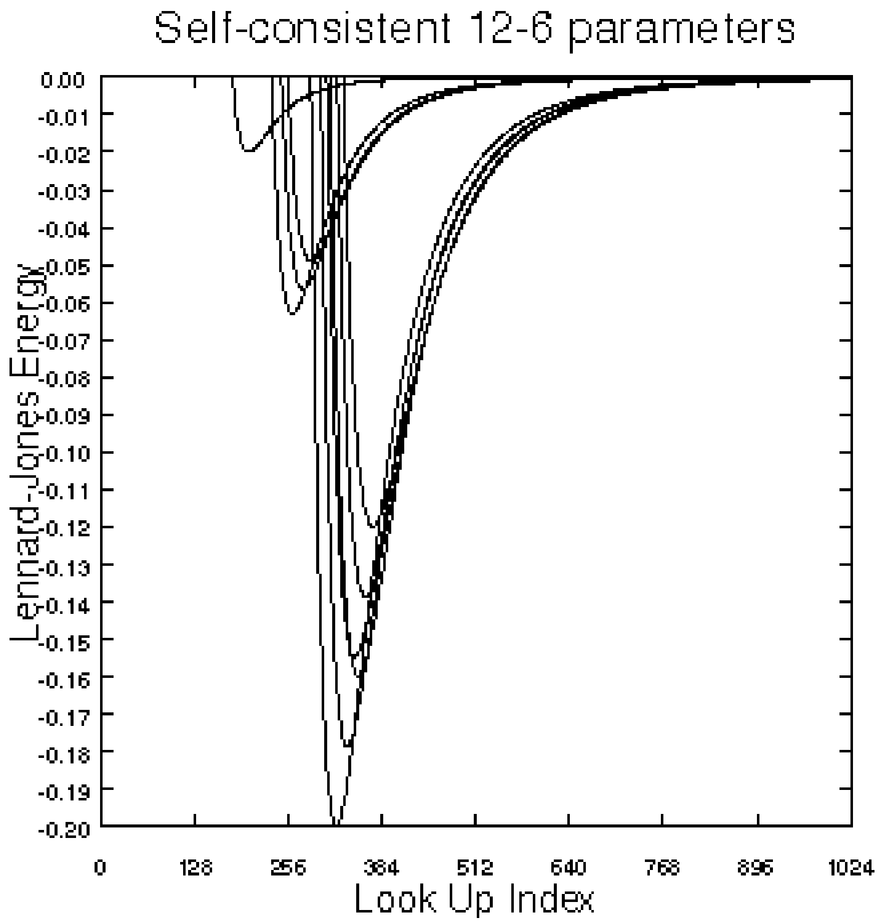
| AMBER atom type | $r_{eqm}$ / Å | ε / kcal mol$^{-1}$ |
|---|---|---|
| C, C*, CA, CB, CC, CD, CE, CF, CG, CH, CI, CJ, CM, CN, CP | 1.850 | 0.12 |
| C2 | 1.925 | 0.12 |
| C3 | 2.000 | 0.15 |
| CH | 1.850 | 0.09 |
| CT | 1.800 | 0.06 |

Using the equations describing $C_{12}$ and $C_6$ above, the following new set of 12-6 parameters were calculated shown in Table 2. These parameters may be used with **AutoDock** version 3.0, or alternatively, you may use or derive your own. Remember the linear regression coefficients for the van der Waals term has not been applied to the parameters in this table.

**Table 2: Self-consistent Lennard-Jones 12-6 parameters, before multiplication by the free energy model coefficients.**

| Atoms i-j | $r_{eqm,ij}$ / Å | $\varepsilon_{ij}$ / kcal mol$^{-1}$ | $C_{12}$ / kcal mol$^{-1}$Å$^{12}$ | $C_6$ / kcal mol$^{-1}$Å$^6$ |
|---|---|---|---|---|
| C-C | 4.00 | 0.150 | 2516582.400 | 1228.800000 |
| C-N | 3.75 | 0.155 | 1198066.249 | 861.634784 |
| C-O | 3.60 | 0.173 | 820711.722 | 754.059521 |
| C-S | 4.00 | 0.173 | 2905899.052 | 1418.896022 |
| C-H | 3.00 | 0.055 | 29108.222 | 79.857949 |
| N-C | 3.75 | 0.155 | 1198066.249 | 861.634784 |
| N-N | 3.50 | 0.160 | 540675.281 | 588.245000 |
| N-O | 3.35 | 0.179 | 357365.541 | 505.677729 |
| N-S | 3.75 | 0.179 | 1383407.742 | 994.930149 |
| N-H | 2.75 | 0.057 | 10581.989 | 48.932922 |
| O-C | 3.60 | 0.173 | 820711.722 | 754.059521 |
| O-N | 3.35 | 0.179 | 357365.541 | 505.677729 |
| O-O | 3.20 | 0.200 | 230584.301 | 429.496730 |
| O-S | 3.60 | 0.200 | 947676.268 | 870.712934 |
| O-H | 2.60 | 0.063 | 6035.457 | 39.075098 |
| S-C | 4.00 | 0.173 | 2905899.052 | 1418.896022 |
| S-N | 3.75 | 0.179 | 1383407.742 | 994.930149 |
| S-O | 3.60 | 0.200 | 947676.268 | 870.712934 |
| S-S | 4.00 | 0.200 | 3355443.200 | 1638.400000 |
| S-H | 3.00 | 0.063 | 33611.280 | 92.212017 |
| H-C | 3.00 | 0.055 | 29108.222 | 79.857949 |
| H-N | 2.75 | 0.057 | 10581.989 | 48.932922 |
| H-O | 2.60 | 0.063 | 6035.457 | 39.075098 |
| H-S | 3.00 | 0.063 | 33611.280 | 92.212017 |
| H-H | 2.00 | 0.020 | 81.920 | 2.560000 |

The above parameters yield the following graphs, for C, N, O and H atom types; the curves in order of increasing well-depth are: HH << CH < NH < OH << CC < CN < CO < NN < NO < OO:-




Grid maps are required only for those atom types present in the ligand being docked. For example, if the ligand being docked is a hydrocarbon, then only carbon and hydrogen grid maps would be required. In practice, however, non-polar hydrogens would not be modeled explicitly, so just the carbon grid map would be needed, for 'united atom' carbons. This saves both disk space and computational time.

## 8. Modelling Hydrogen Bonds

Hydrogen bonds are frequently important in ligand binding. These interactions can be modeled explicitly in **AutoDock**.

In order to save having two types of hydrogen grid maps, and thus conserve disk space, we normally use ligands with just one type of hydrogen, namely polar hydrogens. Polar hydrogens can be defined here as those bonded to heteroatoms like nitrogen and oxygen, while non-polar hydrogens are bonded to carbon atoms.

If you want to model non-polar hydrogens as well, you would need a separate map for such hydrogens. You could use the atom type code ‘h’ for non-polar hydrogens, and ‘H’ for polar hydrogens. Use 12-6 to distinguish non-polar hygrogens, and 12-10 for polar hydrogens.

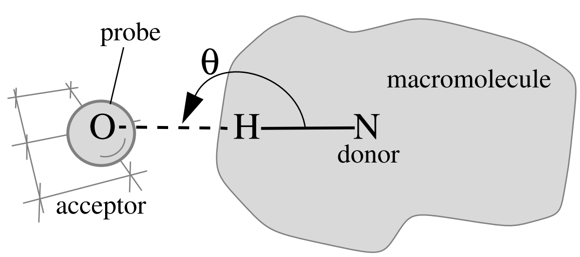
The user *must* specify the appropriate 12-10 parameters in the **AutoGrid** parameter file, and on the correct lines. Pairwise atomic interaction energy parameters are always given in blocks of 7 lines, in the order: C, N, O, S, H, X, M. X and M are “spare” atom types: If there were phosphorus atoms in the receptor, X could be used as P. For example, to model *donor* hydrogens in the ligand, 12-10 parameters would be needed in the *hydrogen* parameter block, but only for H-bond acceptors, N,O and S (second, third and fourth lines in the H-parameters). The other parameters remain as 12-6 Lennard-Jones values (C,H,X and M). In order to keep the symmetry of pairwise energetics (H-O is the same as O-H), the user must specify 12-10 parameters for H (fifth line) in the N, O and S-parameter blocks.

**Table 3: Self-consistent Hydrogen bonding 12-10 parameters, before multiplication by the free energy model coefficients.**

| Atoms i-j | $r_{eqm,ij}$ / Å | $\varepsilon_{ij}$ / kcal mol$^{-1}$ | $C_{12}$ / kcal mol$^{-1}$Å$^{12}$ | $C_{10}$ / kcal mol$^{-1}$Å$^{6}$ |
|---|---|---|---|---|
| N-H | 1.90 | 5.00 | 55332.873 | 18393.199 |
| O-H | 1.90 | 5.00 | 55332.873 | 18393.199 |
| S-H | 2.50 | 1.00 | 298023.224 | 57220.459 |

**AutoGrid** detects hydrogen bond parameters in the grid parameter file, if either *n* is not 12 or *m* is not 6. If so, the pairwise interaction is modulated by a function of the cosine of the hydrogen bond angle. This takes into account the directionality of hydrogen-bonds.

**AutoGrid** incorporates the angular dependence of the hydrogen bond potential. The ideal hydrogen bond would have an angle, θ, of 180˚ between the lone-pair of the acceptor atom, the polar hydrogen and the donor atom, thus:




As θ decreases, the strength of the hydrogen bond diminishes. There are no hydrogen bonds when θ is 90˚ or less.

### *8.1 A note on atom type codes:*

**Note**: If you use hydrogen bonding for nitrogen, you may need to distinguish between nitrogens that can be acceptors and those that can be donors. The above settings for N and H would allow

>NH to accept a hydrogen bond. To avoid this, such nitrogens should be treated as 12-6 non-hydrogen bonders: used ‘n’ as the atom type code instead of ‘N’. This would mean, of course, an extra grid map.

If you use polar and non-polar hydrogens, for example, with atom type codes of ‘H’ and ‘h’, you must edit the atom names in the PDBQ files by hand. This would apply to different flavours of nitrogen, ‘N’ for polar and ‘n’ for non-polar; or carbon, ‘C’ for aliphatic carbons and ‘A’ for aromatic carbons.

# 9. Electrostatic Potential Grid Maps



In addition to the atomic affinity grid maps, **AutoDock** requires an electrostatic potential grid map. Polar hydrogens must be added, if hydrogen-bonds are being modeled explicitly. Partial atomic charges must be assigned to the macromolecule. The electrostatic grid can be generated by **AutoGrid**, or by other programs such as MEAD[14] or DELPHI[15], which solve the linearized Poisson-Boltzmann equation. **AutoGrid** calculates Coulombic interactions between the macromolecule and a probe of charge $e$, $+1.60219 \times 10^{-19}$ C; there is no distance cutoff used for electrostatic interactions. A sigmoidal distance-dependent dielectric function is used to model solvent screening, based on the work of Mehler and Solmajer[16],

$$\varepsilon(r) = A + \frac{B}{1 + ke^{-\lambda B r}}$$

where: $B = \varepsilon_0 - A$; $\varepsilon_0$ = the dielectric constant of bulk water at $25^{\circ}$C = 78.4; $A = -8.5525$, $\lambda = 0.003627$ and $k = 7.7839$ are parameters.

Charges must be stored in ***PDBQ format*** in order for **AutoGrid** to read them. PDBQ is an augmented form of the standard PDB format, in which an extra column is used to store the partial atomic charges (hence the “Q” in “PDBQ”). Columns 71-76 of the PDB file hold the partial atomic charge (the older form of PDBQ contains charges in columns 55-61).

Charges can be assigned using a molecular modeling program. Unix shell scripts are provided to convert from **Insight95**[17] “.car” files (“cartopdbq”) and **SYBYL**[18] “.mol2” files

14. Bashford, D. and Gerwert, K. (1992) “Electrostatic calculations of the $pK_a$ values of ionizable groups in bacteriorhodopsin”, *J. Mol. Biol.*, **224**, 473-486; Bashford, D. and Karplus, M. (1990) “$pK_a$s of ionizable groups in proteins - atomic detail from a continuum electrostatic model.”, *Biochemistry*, **29**, 10219-10225; MEAD is available from Donald E. Bashford, Dept. Molecular Biology, Mail Drop MB1, The Scripps Research Institute, 10666 North Torrey Pines Road, La Jolla, CA 92037.
15. Gilson, M.K. and Honig, B. (1987) *Nature*, **330**, 84-86; DELPHI is available from Biosym Technologies, 9685 Scranton Road, San Diego, CA 92121-2777, USA.
16. Mehler, E.L. and Solmajer, T. (1991) “Electrostatic effects in proteins: comparison of dielectric and charge models” *Protein Engineering*, **4**, 903-910.
17. Biosym/MSI, 9685 Scranton Road, San Diego, California 92121-3752, USA.
18. Tripos Associates, Inc., 1699 South Hanley Road, Suite 303, St. Louis, Missouri 63144-2913, USA.

("`mol2topdbq`"). See also "q.amber" and "q.kollua", in the appendices.

# Methodology

## 10. Getting Started...

This section describes very quickly the method for setting up a docking using the AutoDock programs. You should find all these utilities under the "`share`" and "`bin`" directories. Before you start, add these two lines to your `.cshrc`: "`setenv AUTODOCK_UTI /path/to/the/directory/share`" and "`set path=($path $AUTODOCK_UTI)`". Make sure you "`source .cshrc`" also.

**(1)** The macromolecule first needs polar hydrogens to be added and then partial atomic charges to be assigned. This can be done efficiently in SYBYL, *e.g.*, using the "Biopolymer" menu, adding "Essential_Only" hydrogens and assigning "KOLLUA" partial charges to the protein. Create the PDBQS file[19] for the macromolecule. Save the protein in "mol2" format, and then convert into PDBQS format using "`mol2topdbqs`". This also assigns atomic solvation parameters and creates "`macro.pdbqs`":

```
% mol2topdbqs macro.mol2
```

**(2)** If you already have a PDBQ-formatted version of your macromolecule, say "`macro.pdbq`", you must assign the atomic solvation parameters to it. The "`addsol`" program will input "`macro.pdbq`" and output a PDBQS file, "`macro.pdbqs`":

```
% addsol macro.pdbq macro.pdbqs
```

**(3)** Create the ligand PDBQ file[20] using "deftors"[21], to define any torsions that you want to be explored during the docking. (Label the ligand with "Atom ID" or atom serial numbers in a molecular viewer. This will help in assigning the atoms):

```
% deftors lig.mol2
```

**(4)** Create the GPF (grid parameter file) and the DPF (docking parameter file).

```
% mkgpf3 lig.pdbq macro.pdbqs
% mkdpf3 lig.pdbq macro.pdbqs
```

These create files with names derived from the ligand and macromolecule files,

19. This contains the PDB records in addition to the partial atomic charges and atomic solvation parameters.
20. This contains the root atoms and the branches and torsions defining the rotatable bonds in the ligand, as well as the partial atomic charges.
21. The script `deftors` uses the program AutoTors to assign root atoms and torsions.

namely "`macro.gpf`" and "`lig.macro.dpf`"[22].

**(5)** Edit the GPF and then use AutoGrid to calculate the grid maps.

```
% autogrid3 -p macro.gpf -l macro.glg &
```

**(6)** Edit the DPF and then perform the dockings using AutoDock.

```
% autodock3 -p lig.macro.dpf -l lig.macro.dlg &
```

**(7)** To view docking results in a molecular modelling program, use "`get-docked`", to create a PDB formatted file. It will be called "`lig.macro.dlg.pdb`" and will contain all the docked conformations output by **AutoDock** in the "`lig.macro.dlg`" file.

```
% get-docked lig.macro.dlg
```

Or to view in AVS, use "`mkdlgfld`","`mkatmtypfld`" and "`mkbndfld`".

**(8)** You don't need to do this step. But if you are interested, you can calculate the energy of a given ligand conformation in the crystal structure you used to calculate the maps:

```
% autodock3 -p lig.macro.dpf -l lig.macro.epdb.log -c <
lig.macro.epdb.com
```

where the AutoDock command file "`lig.macro.epdb.com`" contains the two commands, "`epdb lig.pdbq`" and "`stop`" on separate lines.

There are several Unix shell scripts and "`awk`" programs to help set up default parameter files for **AutoGrid** and **AutoDock**. They are described in more detail in the Appendix. The user *must* check their input "gpf" and "dpf" files, to ensure the defaults look reasonable. The user can adjust the default parameters using a text editor like "`vi`" or "`emacs`". These parameters are described in the sections "AutoGrid Parameter File Format" and "AutoDock Parameter File Format", in the appendices.



# 11. Setting Up AutoGrid and AutoDock Jobs

Let us suppose that the user wishes to test **AutoDock** by trying to reproduce an x-ray crystallographic structure of a ligand-enzyme complex taken from the Brookhaven Protein Data Bank. The first step is to split the desired PDB file into two separate PDB files, one containing all the heavy atoms of the enzyme, the other containing those of the ligand. Both files should retain the exten-

22. The stems differ because the grid parameter file is specific to the macromolecule only, but the docking parameter file is specific to both the ligand and the macromolecule. Therefore, try and keep the ligand and macromol filename stems short.

sion '`.pdb`'.

**Note**: Care should be taken when the PDB file contains disordered residues, where alternate location indicators (column 17) have been assigned. For each such atom, the user must select only one of the possible alternate locations (preferably that with the highest occupancy value).

We will discuss in the next sections, the steps needed to prepare the parameter files for **AutoGrid** and **AutoDock**. If desired, the user may specify rotatable bonds in the ligand (receptor flexibility is not allowed). To help this definition, there is a program called **AutoTors**. This utility interactively queries the user about the rigid portion of the molecule (the "root") and rotatable torsions (the "branches" and "torsions"). Then it outputs the ligand in PDBQ format for **AutoDock**. It can even process partial charges on the hydrogens to create a polar-hydrogen only version of the ligand. This will be discussed in greater detail below.

## 12. Preparing the Ligand

Initially you must add hydrogens to all atoms in the ligand, ensuring their valences are completed. This can be done using a molecular modeling package. Make sure that the atom types are correct before adding hydrogens. You may want to specify the pH, depending on whether charged or neutral carboxylates and amides are desired.

Next, assign partial atomic charges to the molecule. AMPAC or MOPAC can be used to generate partial atomic charges for the ligand. These charges must be written out in PDBQ format, which has the same columns as a Brookhaven PDB format, but with an added column of partial atomic charges (see '`cartopdbq`' and '`mol2topdbq`' in Appendix I).

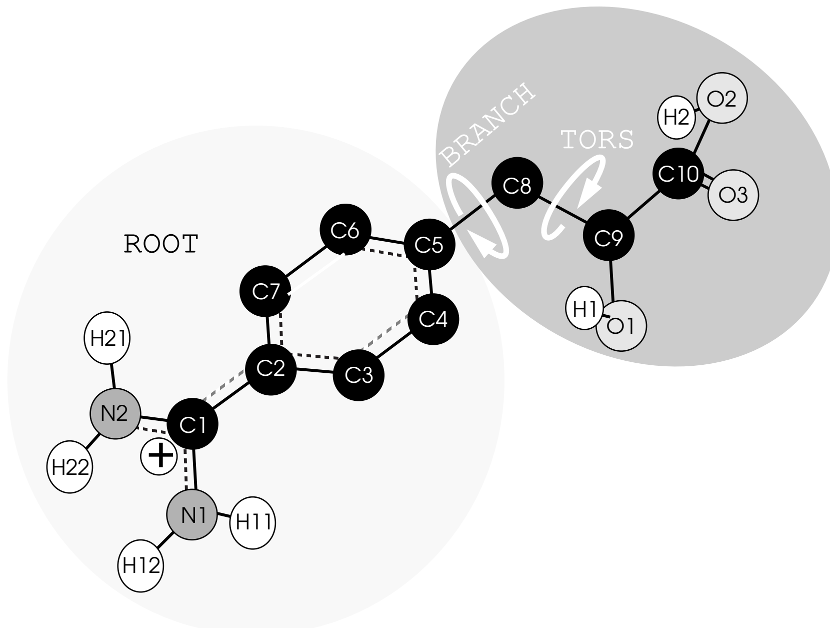
## 13. Ligand Flexibility and Constraints

To allow flexibility in the ligand, it is necessary to assign the rotatable bonds. It is a good idea to have handy a plot of the ligand, labelled by atom name, and a second labelled by atom serial number (atom ID). **AutoDock** can handle up to `MAX_TORS` rotatable bonds: this parameter is defined in "autodock.h", and is ordinarily set to 32. If this value is changed, **AutoDock** must be recompiled.

Torsions are defined in the PDBQ file using the following *tokens or keywords:*

```
ROOT / ENDROOT
BRANCH / ENDBRANCH
TORSION / ENDTORSION
```

These keywords use the metaphor of a tree. See the diagram below for an example. The "root" is defined as the *fixed* portion of the ligand, from which *rotatable* ‘branches’ sprout. Branches within branches are possible, and torsions are a special case of branches, where the two atoms at either end of the rotatable bond have only *two* nearest neighbors (unlike branches which can have three or more). Nested rotatable bonds are rotated in order from the “leaves” to the “root”.




The PDBQ keywords must be carefully placed, and the order of the ATOM or HETATM records may need to be changed in order to fit into the correct branches. The PDBQ keywords can be abbreviated to no less than the first 4 letters. To assist the user in placing these keywords correctly, and in re-ordering the ATOM or HETATM records in the ligand PDBQ file, it is best to use the interactive program **AutoTors** (see below).

Note: **AutoTors**, **AutoGrid** and **AutoDock** do not recognize PDB “CONECT” records, neither do they output them.

“CONSTRAIN” defines a single, optional distance constraint, between two flexible parts of the ligand. It is not normally used in docking. This retains only those conformations where this distance is within a certain range of values. In docking, a conformation which violates this constraint is instantly rejected; it does not increment the rejections-counter in simulated annealing, its energy is not evaluated, nor is the steps-counter incremented. This PDBQ keyword has the following syntax:

```
CONSTRAIN atom1 atom2 lower upper
```

The first two parameters are the atom serial numbers of the two atoms to be constrained, and the last two are the lower and upper bounds for this distance, in Angstroms. This can be particularly useful when docking say two proteins: a loop from one protein can be cut out and the ends constrained to have roughly the same value as in the original protein.

The next sections describe the input files needed for **AutoTors**, and how to run it.

# 14. Using AutoTors to Define Torsions in the Ligand

This section describes input and output files used and generated by **AutoTors**. Input consists of one or two files, depending on whether the ligand is in our "**AutoDock**-standard" PDBQ-format, or in Sybyl's `mol2`-format. PDBQ-format is the default; `mol2`-format is allowed with the "`-m`" flag (see below).

## 14.1 Ligand is in PDBQ-format:

When the ligand is in PDBQ format, AutoTors also needs a "bnd" or bond file, which describes the connectivity of the atoms in the ligand. In this example, the bond file is "oligo.bnd", and "oligo.pdbq" is the input PDBQ file; "oligo.out.pdbq" is created and contains all the ROOT, BRANCH and TORS keywords needed to define the torsions selected by the user.

```
% autotors oligo.bnd oligo.pdbq oligo.out.pdbq
```

The ".bnd" file, contains information about the covalent bonds in the ligand. The bonds are described by the serial numbers of the atoms in the input PDBQ file, with one line per bond. For example, if C10 is the atom appearing on the first "ATOM" line in the PDBQ file, and it is bonded to N18 which appears on the 17th line in the PDBQ file, this information appears as a discrete line in the ".bnd" file as: "1 17". The output of "pdbtoatm" is an "atm" file, which can be converted to a "bnd" file, using "atmtobnd". For example, to generate a "bnd" file, use something like this command:

```
% pdbtoatm vitc.pdbq | atmtobnd > vitc.bnd
```

## 14.2 Ligand is in Mol2-format:

When the ligand is in SYBYL-mol2 format, no "bnd" bond file is required, in addition to `-m` flag. This is because the mol2 file contains both atom coordinates and bonding information. So, for example the following command would read in the "lead.mol2" file and after interactively requesting which torsions to rotate, AutoTors would write out "lead.out.pdbq":

```
% autotors -m lead.mol2 lead.out.pdbq
```

## 14.3 AutoTors Output:

The output filename is defined by the last **AutoTors** command-line argument. Output consists of PDBQ-formatted lines, rearranged as required by **AutoDock,** according to the user's specification of the fixed ROOT portion of the molecule, and the allowed rotatable bonds in the rest of the molecule. **AutoTors** inserts the ROOT, ENDROOT, BRANCH, ENDBRANCH, TORS, and ENDTORS lines in the necessary places.

## 14.4 AutoTors Flags:

**-m <input_ligand_mol2_file>**

This flag is used when the input file is in Tripos 'mol2' format (produced by SYBYL). When it is entered on the command line, the program uses only 1 file for both kinds of input (the bond data input file and the pdbq data input file) and uses the second file specified for output. [If the user runs the program with the -m flag AND three file parameters on the command line, the first file will be opened for reading the input needed by the program, the second opened for writing and the third ignored. This means any contents in the second file will get over-written and lost.]

**`-h`**

This flag causes the program to detect non-polar hydrogens, that is hydrogen atoms bonded to carbon atoms, to merge the charge of each with the charge of the carbon to which it is bonded and to delete the line of output data pertaining to that hydrogen. At the end of the program, a count of the number of non-polar hydrogens which have been merged in this fashion is written to the screen.

**`-o`**

This flag is used only in conjunction with the -h flag when the pdbq data is in the older pdbq format. It causes the program to obtain charge data from column 55 instead of column 70. (Its use along with the -m flag is an error but this is disregarded.)

**`-a`**

This flag instructs AutoTors to disallow torsion rotations in amide and peptide bonds, (C=O)--(NH).

**`-b`**

This flag is useful for peptides. It disallows rotations in backbone torsions, including phi, psi and omega (peptide) torsions.

**`-c`**

This will add atom connectivity to the ATOM records in the output pdbq file.

**`-e`**

This instructs AutoTors to use the atom types given in the mol2file. This can only be used with the '-m' mol2-format flag.

**-r**

This sets the ROOT to be the non-hydrogen atom closest to the center of the molecule.

**-M**

This instructs AutoTors to use the ROTATABLE_BOND and ANCHOR information in a Tripos SYBYL mol2 formatted file, to define the ROOT and active torsions.

**-A**
**-A +<angle>**

This flag causes the program to check rings for aromaticity. If all the ring atoms are 'co-planar' enough, the program replaces 'C' by 'A' for all carbons in the ring. This distinction is necessary in the AutoDock 3.0 Force Field computations. By default, the test for planarity is whether the angle between two adjacent atoms' normal vectors is less than or equal to 7.5 degrees.

The user can specify what cut-off to use (and thus override the default of 7.5 degrees) by typing in a different angle after the -A flag. The angle should be given in degrees. Note: this number must be preceeded by a plus sign, '+'. For example:

```
% autotors -A +6.2 myfile.bnd myfile.pdbq myfile.out
```

would cause autotors to use 6.2 degrees for this aromaticity cut-off angle between adjacent atoms. This depends on how 'warped' the ring is: some crystal structures can have aromatic rings that are quite distorted from planarity.

The `-m`, `-h` and `-a` flag may appear in any order. The `-o` flag must be given after the `-h` flag. Placement of these flags should follow these two examples. Square brackets denote optional flags:

```
% autotors [-h][-o][-a][-c] pept.bnd pept.pdbq pept.out.pdbq
```

For SYBYL-mol2 input, *e.g.*:

```
% autotors -m [-h][-o][-a][-c] drug.mol2 drug.out.pdbq
```

## 15. Running AutoTors

`MAX_TORS`: **AutoDock** is set up to allow a maximum number of torsions. If **AutoTors** detects more torsions than are permitted, a warning to that effect is given and it is up to the user to reduce the number of torsions, either by deleting or selecting the appropriate number of torsions. `MAX_TORS` is defined in the file "`autodock.h`"; if this definition is changed, the autodock-executable must be re-made, using the appropriate Makefile.

There are four stages in running **AutoTors**:

## 15.1 Input Stage

(a) Data about bonds in the molecule are used to construct a tree -like structure. Each line of bond data consists of two integers corresponding to the line numbers (in the pdbq file) of the atoms involved. These integers are used as the 'id's of the atoms in the molecule. Once the new ids are read in, the pre-existing TREE is searched for ATOM_NODEs with either of these ids. If only one such node is found, an ATOM_NODE is created for the other id and linked to the already-entered node in the appropriate way. If the id of the pre-existing node was the first of the two integers on the line of bond data, one of the node's next links is set to the new node and one of the new node's prev links is set to the pre-existing node. In the other case, the opposite linking pattern is set. If neither id can be found in the member ids of ATOM_NODEs in the TREE, two new ATOM_NODEs are created, linked together in the appropriate way and held in a temporary data structure for later linking into the 'TREE' in phase b. If both of the ids are already in the TREE, the two nodes with these ids are linked as appropriate. Moreover, this case signals the detection of a cycle and the existing TREE is processed to detect the members of this cycle and to store resulting information about the new cycle. The members of new cycle i are stored in two dimensional global array cycle[i]. The number of members of cycle i is stored in cycle_size[i].

(b) After all the bond data is entered, the program attempts to attach any pairs which had not been linked to pre-existing atoms. To do this, it searchs through the TREE AT MOST once for each id in input data, attempting to add each unattached pair to the TREE. If any unattached pair remains after this process, the input data is flawed and the program exits early with an error message to that effect.

## 15.2 Root Specification Stage

After all the nodes are created and connected, in whatever order and direction the bond data specifies, the user interacts with the program to select the portion of the molecule to be considered the 'ROOT.' This is the section of the molecule which will remain rigid and NOT undergo any torsions. This phase has two parts:

(a) Cycles detected are listed on the screen. The user either selects one of these cycles to be the root (by entering the appropriate number) OR selects none of them as the root (by entering '0').

(b) The user can modify the list of root atoms at this point by adding atoms to the rootlist (by entering the atom 'id.'). The user leaves this phase by entering 'q' (to quit).

If no root atoms are specified, a message to this effect is written to the screen and the program will exit at this point.

## 15.3 Torsions Detection and Selection Stage

Once at least one root atom has been designated, the program next processes the TREE, changing the direction of the links in the TREE as appropriate (so that all links previous to the root are previous and all nexts are nexts), and accumulating a list of possible torsions which the user edits interactively.

(a) The TREE is traversed in a depth-first order traversal at this point detecting possible torsions. Torsions cannot occur between root atoms NOR between atoms in a cycle. Moreover, torsions are not permitted between atoms and their 'leaves' (attached atoms which have no other connection). In the case of the user-specified '-a' flag, amide bond torsions are not permitted. During this traversal, a linked list of possible torsions is built.

(b) The user is given the list of the torsions detected in the molecule and has an opportunity to modify this list. Torsions can be deleted OR selected at this point.(Depending on whether only a few are to be deleted or if only a few are to be selected). The user leaves this phase by entering 'q' (for quit).

### 15.4 Root Expansion and Output Stage

The rest of program follows at this point with no further input from the user. The TREE is traversed again and the rootlist is expanded to include any atoms which are between the existing root section and 'BRANCH'es as defined by active torsions. Next, the TREE is traversed and new_id numbers are assigned sequentially to the atoms in the TREE, starting with the root. Finally, a last traversal through the TREE is made and output is written to the file specified by the user on the command line. 'REMARK' lines are written first describing each possible torsion and its status at the end of the program. Next, the expanded list of root atoms is output preceeded by a'ROOT' line and followed by an 'ENDROOT' line. Last, the rest of atoms in the molecule are output with appropriate 'BRANCH', 'TORS', 'ENDBRANCH' and 'ENDTORS' lines inserted as dictated by active torsions.



## 16. Adding Polar Hydrogens to the Macromolecule

When modeling hydrogen bonds explicitly, it is necessary to add polar hydrogens to the macromolecule. Then the appropriate partial atomic charges must be assigned. This can be achieved by the user's preferred method, *e.g.* using **InsightII**, **Quanta**, **Sybyl**, **AMBER** or **CHARMm**. Alternatively, one of the shell scripts described in the Appendix can be used. The charged macromolecule must be converted to PDBQS format so that **AutoGrid** can read it.

Note that most modeling systems add polar hydrogens in a default orientation, typically assuming each new torsion angle is 0° or 180°. Without some form of refinement, this can lead to spurious locations for hydrogen-bonds. One option is to relax the hydrogens and perform a molecular mechanics minimization on the structure. Another is to use a program like "pol_h" which takes as input the default-added polar hydrogen structure, samples favorable locations for each movable proton, and selects the best position for each. This "intelligent" placement of movable polar hydrogens can be particularly important for tyrosines, serines and threonines.

# 17. Running AutoGrid

**AutoGrid** requires an input grid parameter file, which usually has the extension ".gpf". The command is issued as follows:

```
% autogrid3 -p macro.gpf -l macro.glg &
```

where '-p macro.gpf' specifies the grid parameter file, and '-l macro.glg' the log file output during the grid calculation. The '`&`' ensures that the this job will be run in the background. This whole line can be prefixed with the 'nice' command to ensure other processes are not unduly affected. The log file will inform the user of the maximum and minimum energies found during the grid calculations.

**AutoGrid** writes out the grid maps in ASCII form, for readability and portability; **AutoDock** expects ASCII format grid maps. For a description of the format of the grid map files, see the appendices.

Check the minimum and maximum energies in each grid map: these are reported at the end of the AutoGrid log file (here, it is "`macro.glg`"). Minimum van der Waals' energies and hydrogen bonding energies are typically -10 to -1 kcal/mol, while maximum van der Waals' energies are around $+10^5$ kcal/mol. Electrostatic potentials tend to range from around $-10^3$ to $+10^3$ kcal/mol: if these are both 0, this is a fairly clear indication that there are no partial charges on the macromolecule.

As well as the grid maps, **AutoGrid** creates two AVS-readable files, with the extensions '.fld', and '.xyz'. The former is a *field file* summarizing the grid maps, and the latter describes the spatial extent of the grids in Cartesian space. (To read the grid maps into AVS, use a "read field" module.)

The '-o' flag can be used on the **AutoGrid** command line to signify that the '.pdbq' file specified in the grid parameter file is in 'old' PDBQ format (charges are stored in columns 55-61).

# 18. Flexible Docking with AutoDock

As already described in the Introduction, **AutoDock** can use *Monte Carlo* simulated annealing (SA), a genetic algorithm (GA), a hybrid genetic algorithm-local search (LGA), an evolutionary programming (EP) or a pure local search (SW or pSW) engine in order to explore the conformational states of a flexible ligand.

Quaternion rotations[23] have been implemented in handling the rigid body orientation of the

23. Shoemake, K. (1985) "Animating Rotation with Quaternion Curves" *SIGGRAPH '85*, **19**, 245-254.

ligand. It was found that this gave finer control over the movement of the ligand, and gave better docked solutions than with the alternative Eulerian rotations. Quaternions also avoid the gimbal lock problem that Eulerian angles suffer from.

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. A simnulated annealing (SA) run is a sequence of constant temperature annealing cycles. A genetic algorithm (GA, LGA or EP) run consists of a series of generations. Each job can be seeded with a user-defined or a time-dependent random-number generator seed. If time-dependent seeds are requested, this value is updated each time a run starts, so 10 runs in one job get 10 different seed values.

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). These flags will be discussed in greater detail later on. It is advisable to do a short run to check the DPF, before committing to spending billions of computer cycles. If there is any problem, a short run should find it.

Whatever search engine is chosen, the DPF must define the following: the random number generator seed or seeds using "`seed`"; the atom "`types`" in the ligand, that match the grid maps produced by **AutoGrid**; the "`fld`" field file that describes the spatial extents of the grids; and the names of the "`map`" files themselves. **AutoDock** must be told what filename contains the ligand to "`move`", and "`about`" which x,y,z coordinate the rotations and translations will be centered. The x,y,z values used in the "`about`" command must be in the same coordinate frame as the coordinates in the ligand PDBQ file specified in the "`move`" command.

Currently in **AutoDock 3.0**, the intial state of the ligand can only be set using SA. All evolutionary search methods, GA, LGA and EP, automatically start with a random population. It is not possible to seed a population with user-defined individuals in version 3.0

The initial translation and quaternion of this ligand may be set in SA dockings only, using the "`tran0`" and "`quat0`" keywords.

The step sizes for making changes to the state variables affect SA *and* the evolutionary methods, GA, LGA and EP. They are defined using the "`tstep`", "`qstep`" and "`dstep`" keywords. The default values are: translation, 0.2 Å, rigid-body orientation and dihedral angles, 5°.

If the ligand is conformationally flexible, the user may specify, for SA only, the number and initial values of the initial dihedral angles using "`ndihe`" and "`dihe0`". If the keyword "random" is given instead of explicit values, the ligand starts the SA with a random conformation.

The internal non-bonded potential parameters are defiend using the "`intnbp_coeffs`" or "`intnbp_r_eps`" keywords. The former accepts coefficients while the latter accepts equilibrium separations in angstroms and well depths in kcal/mol. The latter input method is more intuitive.

The user should specify the level of output during dockings, using "`outlev`". Essentially, the higher this integer, the more output is generated. A value of 1 is normally used.

If the user gives the "`analysis`" command, then after all the docking runs are completed in a given job, cluster analysis or 'structure binning' will be performed. This is based on positional root mean square deviation of corresponding atoms, ranking the resulting families of docked conformations in order of increasing energy. AutoDock writes out a histogram showing the number of conformations in each cluster, and represents it 'graphically' using a bar chart of '`#`' symbols. Search the AutoDock log file for the phrase '`HISTOGRAM`' all in upper-case, and you will see the cluster analysis results.

The default method for structure binning allows for symmetry rotations. For example, a tertiary butyl can be rotated by +/-120° and it will be chemically equivalent to the original conformation. In other cases it may be desirable to bypass this similar atom type checking and calculate the rms on a one-for-one basis: this can be done using the "`rmsnosym`" keyword. When clustering the conformations, the root mean square deviation tolerance "`rmstol`" and reference structure "`rmsref`" filename should be specified. Typical values for `rmstol` range from 0.5 to 1.5 Å.

AutoDock's analysis tool compares all the docked conformations with one-another, and if two conformations have an rmsd that is less than the rmstol value, they are both stored in the same cluster. This is repeated for all conformations, and the clusters are output ranked in order of increasing energy from most negative to most positive. To perform the cluster analysis, the keyword "analysis" must be given after the dockings have finished, on the last line of the DPF. It uses the '`rmstol`', '`rmsref`' and '`rmsnosym`' commands set earlier in the DPF.

The next sections describe the parameters specific to the different search engines.

# 19. *Monte Carlo* Simulated Annealing

During each constant temperature cycle of *Monte Carlo* simulated annealing, random changes are made to the ligand's current position, orientation, and conformation, if flexibile. The new state is then compared to its predecesso. If its new energy is lower than the previous, this new state is immediately accepted. However, if the new state's energy is higher than the last, it is accepted probabilistically. This probability depends upon the energy and cycle temperature (see the first equation in Section 2). Generally speaking, at high temperatures, many states will be accepted, while at low temperatures, the majority of these probabilistic moves will be rejected.

The user can choose whether to select the minimum energy state found during a cycle to be used as the initial state for the next cycle, or the last state. The best docking results tend to be achieved by selecting the minimum energy state from the previous cycle.

The initial annealing temperature "`rt0`" should be of the order of the average $\Delta E$ found during the first cycle. This ensures that the ratio of accepted to rejected steps is high at the start. A typical automated docking job may have an initial annealing temperature "`rt0`" of 500 (depending on the system's average $\Delta E$) and a temperature reduction factor "`rtrf`" of 0.85-0.95 /cycle. Gradual cooling is recommended, to avoid "*simulated quenching*", which tends to trap systems in local minima.

Depending on the degree of complexity of the problem, a relatively good search is given by 50 *Monte Carlo* "`cycles`", and a maximum of 30,000 steps rejected "`rejs`" or 30,000 steps accepted "`accs`". 10 "`runs`" may or may not give a range of possible binding modes,. Multiple runs also give relative energies. A schedule of 100 runs, 50 cycles, 3,000 steps accepted, 3,000 steps rejected will provide more highly populated clusters, hinting at the 'density of states' for a given conformation. A short test job would be: 1 run, 50 cycles, 100 accepted, 100 rejected steps.

The user must specify the maximum step a state variable can make in one step. Furthermore, these can be adjusted during *Monte Carlo* simulated annealing, if a reduction factor (a fraction from 0 to less than 1) for translations and rotations is given. At the start of each cycle, the range from the previous cycle is multiplied by this constant to give the new range, for translational and angular displacements.

If desired, the states can be sampled during a docking and output to a trajectory file. This file contains all the state variables required to define each sampled conformation, position and orientation of the ligand. The user can specify the range of cycles to be sampled. This allows the selection of the last few cycles when the docking will be nearing the final docked conformation, or the selection of the whole run.

## 20. Genetic Algorithm and Evolutionary Programming Docking

Since the new search methods were implemented in an object-oriented fashion, there is a new way of specifying the parameters, that the user should be aware of. All the relevant parameters should be specified first. Then, in order to use the genetic algorithm, the user must set up a global optimizer object using the "`set_ga`" command. Otherwise, if this `set_ga` command is given before the "`ga_*`" parameters are specified, AutoDock will ignore these GA parameters and use the default GA object, which has default parameters built-in.

Both the GA and LGA begin with a population of random ligand conformations in random orientations and at random translations. The user must decide the number of individuals in the population, using "`ga_pop_size`": we have typically found 50 to be a good value. AutoDock counts the number of energy evaluations and the number of generations as the docking run proceeds: the run terminates if either limit is reached ("`ga_num_evals`" and "`ga_num_generations`" respectively). The user can set the number of the best individuals in the current population that automatically survive into the next generation, using "`ga_elitism`": typically this is 1. The user can specify the rate of gene mutation using "`ga_mutation_rate`" and the rate of gene crossover "`ga_crossover_rate`"; typically these are 0.02 and 0.80 respectively, although setting "`ga_crossover_rate`" to 0.00 reduces the genetic algortihm (GA) to an evolutionary programming (EP) method. If the EP approach is used, you should also use an increased mutation rate to ensure a good exploration of the search space. The number of generations for picking the worst individual is set by "`ga_window_size`" and is usually 10.

If the user wants to perform a local search (LS), and for the Larmarckian GA (LGA), the user must specify the local search parameters first (`ls_*`), and then set them (`set_sw1` or `set_psw1`).

The maximum number of local search iterations is set by “sw_max_its”: this is typically about 300. The maximum number of consecutive successes or failures are both typically 4, and should be set by “sw_max_succ” and “sw_max_fail” respectively. The size of the local search space to sample is set by “sw_rho” and is usually 1.0. The lower bound on rho, “sw_lb_rho”, sets the smallest step size that a move can make before terminating the local search, and is usually 0.01. The probability that an individual in the population will experience local search is set by “ls_search_freq”, and is typically about 0.07.

After specifying the local search parameters using the “ls_*” keywords, the user must set up a local optimizer object using “set_sw1” or “set_psw1”, for Solis and Wets or pseudo Solis and Wets. The former is the standard implementation of the local search, while the latter allows the variances which control a step’s size to differ from gene to gene. This latter method, pseudo-SW, is preferable in docking, since a 1 Å-step in translational space is small in comparison to a 1 radian-step in rotation space. The pseudo-SW local search takes its cue about the relative sizes of the translational, orientational and torsional step sizes from the tstep, qstep and dstep values set earlier in the AutoDock input parameter file.

Having set all these parameters and made all these choices, the user must tell AutoDock what to do.

To perform a number of local searches (or “energy minimizations”) the user should put this line into the DPF: “do_local_only 50”, where the number after the command is the number of local search dockings to perform.

To carry out simulated annealing dockings, the command “simanneal” should be given; this will perform the number of runs set by the “runs 10” command, which here would be 10 runs.

To do a number of standard genetic algorithm (GA) dockings, give the “ga_run 10“ command, but do not use the “set_sw1” or “set_psw1” commands in the same DPF. In this example, AutoDock would do 10 GA dockings.

To use the Lamarckian genetic algorithm (LGA) in dockings, you must still use the “ga_run 10” command, but you must have specified either the “set_sw1” or “set_psw1” command in one of the preceding lines of the DPF.

Finally, after the docking command, you will almost certainly want to perform cluster analysis on your search results. Give the ’analysis’ command, and after the last docking run is completed, AutoDock will perform conformational clustering and then output a histogram ranked by increasing energy.

## 21. Running AutoDock

Once the grid maps have been prepared by **AutoGrid** and the docking parameter file, or DPF, is ready, the user is ready to run an **AutoDock** job. A docking is started from the command line using the following command:

```
% autodock3 [-o][-k][-i][-u][-t] -p lig.macro.dpf [-l
lig.macro.dlg] &
```

Input parameters are specified by "`-p lig.macro.dpf`", and the log file containing the output and results from the docking is defined by "`-l lig.macro.dlg`". This is the normal usage of **AutoDock**, and performs a standard docking calculation.

**-o**

This can be added to the command line, to signify that the input file specified in the docking parameter file is in *old* PDBQ format, with charges in columns 55-61.

**-k**

*keep* the original residue number of the input ligand PDBQ file. Normally **AutoDock** re-numbers the starting position to residue-number 0, and any cluster-representatives are numbered incrementally from 1, according to their rank (rank 1 is the lowest energy cluster).

**-i**

This is used to *ignore* any grid map header errors that may arise due to conflicting filenames. This overrides the header checking that is normally performed to ensure compatible grid maps are being used.

**-u**

This returns a message describing the command line *usage* of **AutoDock**.

**-t**

This instructs **AutoDock** to parse the PDBQ file to check the *torsion* definitions, and then stop.

The Unix script "**job**" can be used to submit an **AutoDock** job, and then perform additional post-processing, such as profiling, extracting job-information and creating a field file for AVS display of the docked results. See the Appendix for more details.

## 22. Using the Command Mode in AutoDock

**AutoDock** can be run in "command mode", using the "**-c**" flag thus:

```
% autodock3 -p lig.macro.dpf -l lig.macro.clg -c
```

When **AutoDock** has read in the grid maps specified in "`lig.macro.dpf`", the program gives the message "COMMAND MODE" and waits for the user to issue a command from the *standard input*. These commands are described in more detail below.

An alternative way of using the command mode is to edit a file containing the commands you wish **AutoDock** to execute (say "`command.file`") and channel the output to a file (say "`command.output`"), thus:

```
% autodock3 -p lig.macro.dpf -l lig.macro.clg -c < command.file
> command.output &
```

**AutoDock** can also be used in a UNIX pipe command. This is valuable when an alternative search procedure is desired. Here, the alternative search procedure issues commands to the standard output, and reads the results from the standard input. In this case, **AutoDock** is behaving as an energy server for the alternative search-procedure program.

There are eight recognized commands: **AutoDock**'s command interpreter is not case sensitive.

| | |
|---|---|
| "`eval`" | Evaluate this state's total energy. |
| "`epdb`" | Evaluate the energy of the named PDBQ file. |
| "`outc`" | Output the last state's PDB-formatted Cartesian coordinates. |
| "`oute`" | Output (non-bond and electrostatic) energy breakdown, by atom. |
| "`traj`" | Convert an SA trajectory file into PDB-formatted Cartesian coordinates. |
| "`stop`", "`exit`", "`quit`" | Stop the AutoDock command mode interpreter. |

## *eval*

Evaluates the total energy of a state defined by the subsequent state variables. This command utilizes the trilinear interpolation routine in **AutoDock** along with the supplied grid maps defined in the parameter file specified after the '`-p`' flag to return this energy. The internal energy of the ligand is also taken into account, as dictated by the values of the torsion angles supplied in the *ntor* lines following this `eval` command line; *ntor* is the number of torsion angles defined in the ligand `PDBQ` file, as described in the section "Defining Torsions in AutoDock". The usage of this command is:

`eval <float> <float> <float> <float> <float> <float> <float>` *}* $T_x, T_y, T_z, Q_x, Q_y, Q_z, Q_w$ *(in °)*
`<float>` *}* $i^{th}$ *torsion angle, in °.*
:
: *ntor* lines.

where: $T_x$, $T_y$, $T_z$ are the coordinates of the center of rotation of the ligand; $Q_x$, $Q_y$, $Q_z$ is the unit vector describing the direction of rigid body rotation, about which a rotation of angle $Q_w$ degrees will be applied. The following *ntor* lines hold the torsion angles in degrees, given in the same order as described in the **AutoDock** log file.

### ***epdb***

Calculates the energy of the molecule provided in the PDBQ file, thus:

```
epdb lig.pdbq
```

where: "`lig.pdbq`" is the PDBQ formatted coordinates of a molecule for which the interaction energy with the macromolecule will be returned. The '`-o`' flag supplied at the **AutoDock** execution line specifies the old format of PDBQ, with charges in columns 55-61; otherwise it is assumed that the charges are in columns 71-76.

This command is useful when the state variables for a given molecule are not known, e.g. the x-ray crystallographic conformation of the ligand.

### ***outc***

Returns the coordinates of the ligand at its current transformed position (in the form of a PDB REMARK). The *x*,*y*,*z* coordinates will be determined by the state variables supplied to the `eval` command.

### ***oute***

Returns the total internal energy of the ligand and the total energy of the complex, at the current state variables. These two REMARK lines are written in PDB format, to the command output channel and the log file.

### ***traj***

Convert a simulated annealing "`.trj`" file into PDBQ format. Usage:

```
traj lig.trj
```

where "`lig.trj`" is a simulated annealing (SA) trajectory file written out by an earlier run of **AutoDock**. This trajectory file contains the state variables for the states sampled during a simulated annealing docking simulation. The torsions are assumed to be in exactly the same order as the input ligand PDBQ file. The torsion angles in the trajectory file are *relative* to the input ligand's conformation.

See also the Appendix, script "`runtrj`"; and the next section, "Trajectory Files".

### ***stop, exit, quit***

Halts the execution of AutoDock. A value of 0 is returned by the program, and the message "autodock: Successful Completion" is written to the log file and standard error. Timing information is also written. Note: "`stop`", "`exit`" and "`quit`" are synonymous.

# 23. Trajectory Files

*Note*: Trajectories can only be generated during a simulated annealing run: they are not available for the population-based genetic algorithm methods.

A trajectory (of state variables) can be written out during a normal simulated annealing docking , if the trajectory-frequency (set by the keyword "`trjfrq`") in the docking parameter file is greater than zero. This value defines the output frequency, in steps, for states sampled during the run. The default trajectory filename extension is "`.trj`" . These *state variables* are all that is needed to regenerate the coordinates of the ligand. The trajectory control parameter (either "A" or "E") allows the user to record only *accepted* moves (A); or, moves which are *either* accepted or rejected (E). Just for information, a sample "`.trj`" trajectory file is shown below; you will not need to create such files (unless you feel like creating an animation!):

---

```
ntorsions 2
run 1
cycle 1
temp 300.000000
state 1 A -3.745762 -1.432243 -9.518171 23.713793 23.076145 0.713534 -0.023818 0.700216
30.606248
-4.894825
2.661499
:
:
state 6 R -12.679995 -1.452641 -9.259430 21.634645 23.135242 0.653369 -0.440832
0.615448 39.127316
-31.636299
10.261519
state 7 a -8.746072 -1.458231 -9.080998 21.356874 23.325665 0.648312 -0.448577 0.615200
41.075955
-37.935175
11.918847
:
```

---

There are several keywords: "`run`" and "`cycle`" are self-explanatory; "`ntorsions`" is the total number of changing torsions in the ligand; "`temp`" is the annealing temperature for all subsequent entries, unless otherwise stated. Each "state" record has the format:

**`state nstep acc_rej_code e_total e_internal x y z qx qy qz qw`**

where:

`nstep` = the number of the step, within this cycle;

`acc_rej_code` = '`A`' = an accepted move whose energy was **lower** than its previous state;
= '`a`' = an accepted move whose energy was **higher** than its previous state, which nevertheless passed the *Monte Carlo* probability test at

| | |
|---|---|
| | this temperature; |
| | = ‘R’ = a rejected move. |
| | = ‘e’ = an edge-hit, also a treated as a rejected move. |
| `e_total` | = total energy of the system, ligand + macromolecule; |
| `e_internal` | = internal energy of ligand only; |
| `x,y,z` | = translation of ligand center; |
| `qx,qy,qz,qw` | = quaternion, which describes the ligand’s orientation; |

In order to get a coordinate-based trajectory file, for visualization, the command mode of **AutoDock** must be used to regenerate the coordinates from the state variables. Use the “`traj`” command with the name of the pre-calculated trajectory file. For example, suppose there is a command file called “`trj.conv.com`” that contains:

```
traj lig.trj
stop
```

**AutoDock** would be executed a second time using the following command,

```
% autodock3 -p lig.macro.dpf -l lig.macro.trj.conv.log -c <
trj.conv.com > trj.conv.out
```

## 24. Evaluating the Results of a Docking

At the end of an **AutoDock** job in which more than one run was performed, the program outputs a histogram of clusters and their energies. Look in the `lig.macro.dlg` file for the word ‘`HISTOGRAM`’ all in upper-case. The clustering or *structure binning* of docked conformations is determined by the rms tolerance specified in Å by the “`rmstol`” keyword. The best conformation from each cluster (*i.e.* that with the lowest energy) is written out in PDBQ format at the end of the log file.

Use the UNIX `grep` command to extract information from the docking log file. For example,

```
% grep "^DPF>" lig.macro.dlg | sed 's/^DPF> //'
```

would extract all the lines that begin with “`DPF>`”, and pipe them into the stream editor, “`sed`”, to strip out the “`DPF>`” prompts. Since each line read in from the input DPF is echoed in the log file on such lines, this UNIX command would recover the original DPF that was used to generate “`lig.macro.dlg`”.

To extract the conformations from the docking log file just use the “`dockedtopdb`” command:

```
% dockedtopdb lig.macro.dlg > lig.macro.dlg.pdb
```

This writes out a PDB formatted file, and uses the '`MODEL`' and '`ENDMDL`' records to denote the different dockings. Check that your molecular modelling package or viewer can parse these PDB records.

Or if you need a PDBQ formatted file, use "`dockedtopdbq`" thus:

```
% dockedtopdbq lig.macro.dlg > lig.macro.dlg.pdbq
```

These docked structures can be read into any appropriate molecular modeling program, and the results compared, where possible, with the experimental data.

The table of ranked clusters , under the heading 'CLUSTERING HISTOGRAM' in the log file, shows the final docked energy for each conformation, and the rms difference in Å between the lowest energy member of the cluster and every other member. The rms for the lowest member of the group is by definition zero. You can extract this clustering histogram very easily using this command, which will print the results to the terminal:

```
% gethis lig.macro.dlg
```

After this table in the '`lig.macro.dlg`' log file, the docked structures are output in PDBQ format. Each conformation has a set of REMARK records, one of which describes the rms difference between itself and the coordinates specified in the original input PDBQ file.This can be useful for comparing how close each docked conformation is to the experimentally determined position.

## 25. Visualizing Grid Maps

If you have access to **AVS**, you can visualize the grid maps by using a *read field* module. The user must specify the '`.fld`' file that was created by **AutoGrid**, in order to read in the grid maps. An *extract scalar* module selects the grid map of interest, *e.g.* carbon affinity or electrostatics. The resulting grid map data can be analyzed using *arbitrary slicer* and *isosurface* modules, in order to examine cross sections and iso-energy contours respectively. Negative energy contours are most informative for the atomic affinity grid maps, since they reveal favorable regions of binding.

## 26. Visualizing Trajectories

Trajectories can also be read into **AVS** using the *read field* module. The trajectory file is essentially a set of "stacked" or concatenated PDB frames, and must be read in as a two dimensional field (being the number of atoms in the ligand, and the number of frames in the trajectory file). By paging through this field, using the *orthogonal slicer*, continuous replay of the trajectory can be achieved using an *animate integer* module to control which PDBQ frame is selected by the orthogonal slicer. This animates the sequence of sampled states and allows the user to view in real time the progress of the docking simulation.

Unlike AVS, **gOpenMol** is free, and can be dowloaded via the web from:

`http://laaksonen.csc.fi/gopenmol/gopenmol.html`

It has various **AutoDock** tools, including the ability to read in and view AutDock trajectories: see:

`http://laaksonen.csc.fi/gopenmol/help/main_autodock_widget.html`

# Appendix I: Shell Scripts and Awk Programs

## *cartopdbq*

*Usage*: **`cartopdbq lig.car > lig.pdbq`**

Converts from Biosym InsightII ".car" format to PDBQ format

## *check-qs*

*Usage*: **`check-qs lig`**
*Needs:* **`lig.pdbq`**
*Creates:* **`lig.err`**

Checks partial atomic charges in PDBQ file; any non-integral charges are reported.

## *checkqs*

*Usage*: **`checkqs lig`**
*Needs:* **`lig.pdbq`**
*Creates:* **`lig.err`**

Sorts the input PDBQ file by residue number before running the result through `check-qs`.

## *clamp*

*Usage*: **`clamp grid.map > grid.map.NEW`**

Clamps any **AutoGrid** map values that exceed ECLAMP (normally set to 1000.0)

## *cnvmol2topdbq*

*Usage*: **`cnvmol2topdbq lig.mol2 > lig.pdbq`**
*Needs:* **`lig.mol2`**
*Creates:* **`lig.pdbq`**

This converts from (fixed format) Tripos SYBYL mol2 fiormat into PDBQ format, but stores all the residues' chain-IDs specified by the SUBSTRUCTURE records in the mol2 file. These chain-IDs are then output when the PDBQ lines are written.

## deftors

*Usage*: **`deftors lig.mol2`**
*Creates*: **`lig.pdbq, lig.err`**

Sets up rotatable bonds for **AutoDock**. This script launches **AutoTors**, with the -A +15.0, -a, -h and -m flags; it also checks the charges in the output PDBQ file, with **`check-qs`**.

## dpf3gen

*Usage*: **`dpf3gen lig.pdbq > lig.dpf`**

This is normally used by `mkdpf3`, so you should not use this script by hand.

It generates a pre-cursor to a default **AutoDock** docking parameter file. You must edit the file before using it. This reads in the small molecule PDBQ file, detects all atom types present in the lig.pdbq; and creates a docking parameter file for **AutoDock**. Note the user must replace the tags `<lig>` and `<macromol>` by appropriate filename stems.

This uses equilibrium separations and well depths to define pairwise energy potentials, rather than coefficients.

## get-coords

*Usage:* **`get-coords lig.vol > lig.txt`**

This is used as part of `prepare`, `prepare-gpf+dpf`, `prepare II` and `prepare III`. It takes the `.vol` file created by `pdb-volume` and creates a line that can be used in the grid parameter file to specify the center of the maps.

## get-docked

*Usage*: **`get-docked lig.macro.dlg`**
*Creates*: **`lig.macro.dlg.pdb`**

This extracts the docked records from a docking log file. This is very useful when wanting to view the results of a docking in a molecular modelling program or molecular viewer. It is essentially the same as the ‘`dockedtopdb`’ awk program.

## gethis

*Usage*: **`gethis lig.macro.dlg`**

This outputs the histogram from the confomational analysis section of the **AutoDock** log file, `lig.macro.dlg`, and writes it to the screen.

## getready

*Usage*: **`getready lig.pdb`**
*Needs:* **`pdbinfo, pdbsplitchains, pdbwaters, pdbdewater`**
*Creates:* **`lig.info, lig_.atm.pdb, lig_.het.pdb, lig_.wat.pdb`**

This is a very useful script to get started. It will split a PDB file into separate files, each containing a different chain, and will split each of these chains into ATOM, non-water HETATM and water containing PDB files. Also the .info file is a useful summary description of the PDB file.

## gpf3gen

*Usage*: **`gpf3gen lig.pdbq[s] > lig.gpf`**

This is used by `mkgpf3`, and should *not* be used by hand; otherwise the user must edit certain tags by hand before this can be used by **AutoGrid**.

This generates a precursor to a grid parameter file. It takes `lig.pdbq` as its input file, detects all atom types present, and creates the properly formatted parameter file for **AutoGrid**. It uses equilibrium separations and well depths to define pairwise energy potential. It also assigns atomic solvation parameters, based on Stouten, P.F.W., FrÖmmel, C., Nakamura, H., and Sander, C. (1993), "An effective solvation term based on atomic occupancies for use in protein simulations", *Molecular Simulation*, **10**, 97-120.

## histable

*Usage*: **`histable lig.macro.dlg`**
*Creates*: **`lig.macro.dlg.tbl`**

This extracts the histogram from the docking log file, and counts all the '#' symbols, writing the result in a table file. This is suitable for input to a variety of graph drawing programs and spreadsheets.

## job3

*Usage*: **`job3 lig.macro > lig.macro.joblog &`**

Launches a single **AutoDock 3.0** job. It assumes that "`lig.macro.dpf`" exists, and executes **AutoDock** using the arguments:

```
autodock3 -p lig.macro.dpf -l lig.macro.dlg
```

You must edit this script the first time you use it, so that the environment variables $root, $bin and $sh are correctly set equal to, respectively: the path to the root of **AutoDock** tree, the architecture-dependent binary subdirectory and the Unix scripts subdirectory. The file `lig.macro.joblog` contains the output from the job script.

## *makebox*

*Usage*: **`makebox macro.gpf >! macro.gpf.box.pdb`**
*Creates*: **`macro.gpf.box.pdb`**

This creates a PDB file from the grid parameter file 'macro.gpf', that shows how big and where the grid box will be when AutoGrid calculates the grid maps. You can use this 'box molecule' to help refine the center and number of grid points in the grid maps, before you run AutoGrid.

If you colour the 'box molecule' by atom type, *i.e.* red for oxygen, green for carbon, and blue for nitrogen, then the edges of this box will be coloured-coded to indicate the Cartesian axes. R,G,B will correspond to *x,y,z,* respectively. Your molecule viewer must obey the CONECT records in the '`macro.gpf.box.pdb`' file, even though the corresponding bonds may appear too long, otherwise the edges of the grid box will not be displayed.

## *mkbox*

*Usage*: **`mkbox macro.gpf >! macro.gpf.box.pdb`**
*Creates*: **`macro.gpf.box.pdb`**

This is very similar to '`makebox`', except that this puts a *phosphorus* atom at the minimum *x*, minimum *y* and minimum *z* coordinates of the box. This helps to convey which directions are +*x,* +*y* and +*z*. Once again, if oxygen is red, carbon is green and nitrogen is blue, then R,G,B will correspond to *x,y,z,* respectively.

## *mkdlgfld*

*Usage*: **`mkdlgfld lig.macro.dlg`**
*Needs*: **`lig.macro.dlg`**
*Creates*: **`lig.macro.dlg.fld`**

*Only needed for AVS users.*

This extracts the "AVSFLD" records from an **AutoDock** log file, and puts them in lig.macro.dlg.fld. These "AVSFLD" descriptors must be removed before the file can be used in AVS.

## mkdpf3

*Usage*: **mkdpf3 lig.pdbq macromol.pdbqs**
*Needs*: **dpf3gen, dpf3gen.awk** *(AWK program)*
*Creates*: **lig.macro.dpf**

This creates a default docking parameter file for **AutoDock** 3.0; it needs the ligand in PDBQ format and the macromolecule in PDBQS format. It uses the script `dpf3gen`, which in turn calls the `awk` program '`dpf3gen.awk`'. The `lig.macro.dpf` docking parameter file is based on the atom types detected in the input `lig.pdbq` file. See `dpf3gen` above.

## mkgpf3

*Usage*: **mkgpf3 lig.pdbq macromol.pdbqs**
*Needs*: **gpf3gen, gpf3gen.awk** *(AWK program),* **pdbcen** *(AWK program)*
*Creates*: **macro.gpf**

This creates a default grid parameter file for **AutoGrid** 3.0; it needs `gpf3gen.awk` and `pdbcen`, both `awk` programs. See `gpf3gen` above.

## mol2fftopdbq

*Usage*: **mol2fftopdbq lig.mol2 > lig.pdbq**
*Needs:* **lig.mol2**
*Creates:* **lig.pdbq**

Converts from free formatted SYBYL mol2 into **AutoDock** PDBQ format. Chain-IDs specified in the mol2 file by the `SUBSTRUCTURE` records are incorporated into the PDBQ file.

## mol2topdbq

*Usage*: **mol2topdbq lig.mol2**
*Needs:* **lig.mol2**
*Creates:* **lig.pdbq**

Converts from fixed-format SYBYL mol2 into **AutoDock** PDBQ format, and automatically names the output based on the stem of the input mol2 file. Do not use "`mol2topdbq lig.mol2 > lig.pdbq`", because "`lig.pdbq`" is automatically created.

## mol2topdbqs

*Usage*: **mol2topdbqs lig.mol2**

*Needs:* **`lig.mol2`**
*Creates:* **`lig.pdbqs`**

Converts from SYBYL mol2 format into **AutoGrid** 3.0 PDBQS format, by calling `mol2topdbq` then running `addsol` on the intermediate PDBQ file. Like `mol2topdbq`, it also removes any lone pairs (using "`rem-lp`"), and automatically names the output based on the stem of the input mol2 file. There is no need to use "`mol2topdbq lig.mol2 > lig.pdbq`", because "`lig.pdbqs`" is automatically created.

## pdbcen

*Usage*: **`pdbcen lig.pdb`**
*Creates*: *a "gridcenter" line in AutoGrid GPF format, holding the x,y,z coordinates of the molecule.*

This calculates the center of a molecule supplied in PDB format, and outputs a line holding the x,y,z coordinates of the molecule for inclusion in an **AutoGrid** 3.0 grid parameter file (GPF).

## pdb-center

*Usage*: **`pdb-center [ lig.pdb | lig.pdbq ] > lig2.pdb`**

Calculates the center of mass of each residue; writes these coordinates out using REMARK records.

## pdb-center-all

*Usage*: **`pdb-center-all [ lig.pdb | lig.pdbq ] > lig2.pdb`**

Calculates the center of mass of each residue; writes these coordinates out using REMARK records.Also calculates the center of all the residues.

## pdb-distance

*Usage*: **`pdb-distance macro.pdb`**

The first line of the `macro.pdb` file defines the center of the distance profile. It is just a copy of the line containing the atom of interest, which will be the origin for the distance calculations. However, it must have the ATOM or HETATM record replaced with a non-PDB tag, 'FROM'. The x,y,z coordinates in this FROM line will then be used to calculate the distance to the center of each residue in the protein. Finally, this awk program outputs a bar chart using '#' symbols, showing the distance from this point to each residue. This can be useful to identify all the residues nearest a particular ligand atom, or near an active site.

### pdbdewater

*Usage*: **`pdbdewater macro.pdb >! macro.dry.pdb`**

This removes any water records from a PDB file.

### pdbinfo

*Usage*: **`pdbinfo macro.pdb`**

Builds a summary of the contents of a PDB file.

### pdb-volume

*Usage*: **`pdb-volume [ lig.pdb | lig.pdbq ] > lig2.pdb`**

Calculates the center of mass of each residue. Writes out REMARKs showing these coordinates. Draws ASCII diagram showing volume extents of each residue.

### pdbqtobnd

*Usage*: **`pdbqtobnd lig`**
*Needs:* **`lig.pdbq`**
*Creates:* **`lig.bnd`**

Creates “`lig.bnd`” from the existing “`lig.pdbq`” ligand PDBQ file. Note this script needs just the stem of the file name. This script executes “`pdbqtoatm`” and “`atmtobnd`”: the latter is an executable, not a script, so it must be compiled for each architecture and operating system used.

### pdbqtopdb

*Usage*: **`pdbqtopdb lig.pdbq > lig.pdb`**

Converts from **AutoDock** PDBQ to PDB format.

### pdbsplitchains

*Usage*: **`pdbsplitchains macro.pdb`**

Creates separate PDB files that contain each of the chains in `macro.pdb`. The chain IDs are used to name the new PDB files. If there is no chain ID, the underscore character, '_' is used.

## pdbtoatm

*Usage*: **`pdbtoatm lig.pdbq > lig.atm`**

This creates a Connolly ATM formatted file "`lig.atm`" from the ligand PDBQ file, "`lig.pdbq`". This is used to create input for the utility program **`atmtobnd`** to generate a bond connectivity file.

## pdbdewaters

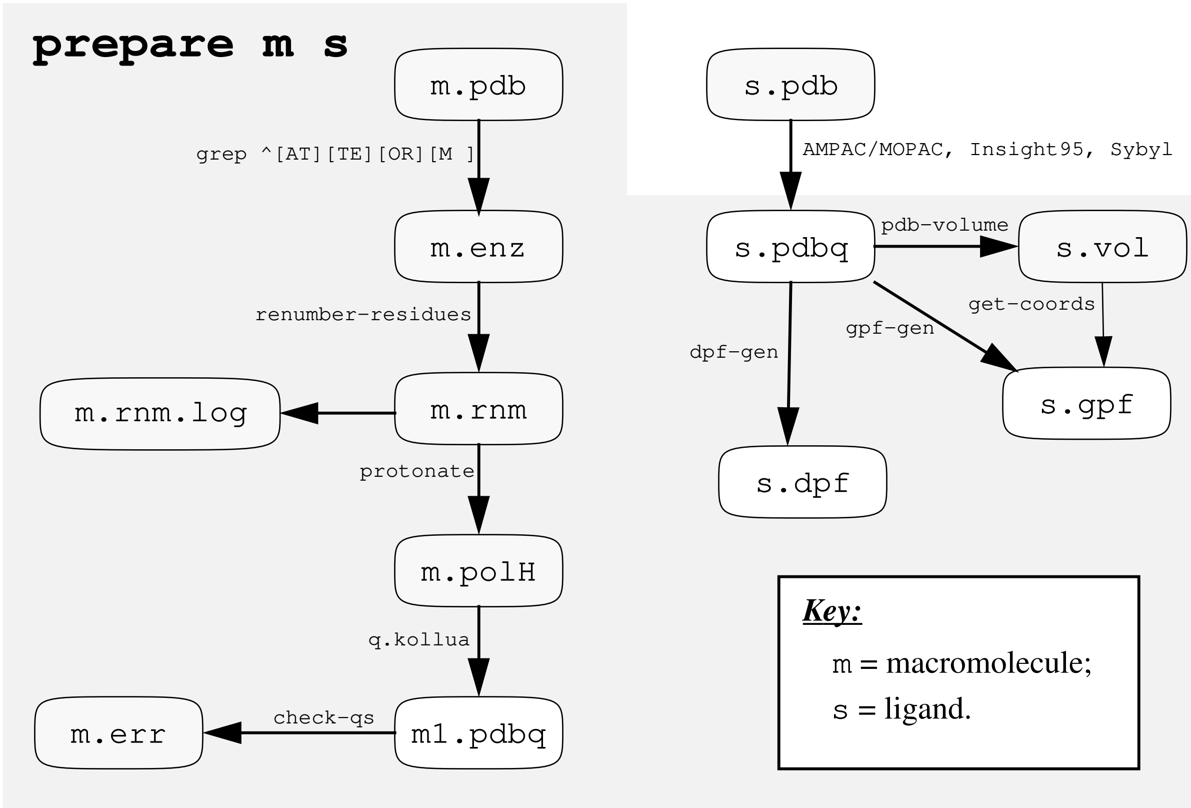
*Usage*: **`pdbwaters macro.pdb > macro.wet.pdb`**

Extracts the waters from `macro.pdb` into `macro.wet.pdb`.

## prepare

*Usage:* **`prepare m s`**

*where*: `m.pdb` and `s.pdbq` contain the receptor and ligand respectively. `Prepare` performs the following eight steps. The macromolecule '`.pdb`' filename stem is represented by "`m`", and the ligand '`.pdbq`' filename stem by "s":




1. Extracts all ATOM and TER records from `m.pdb` into `m.enz;`
2. Renumbers residues to avoid problems in `protonate`-step;
3. Adds polar hydrogens to `m.enz`, creating `m.polH`;
4. Somewhat crudely assigns partial atomic charges to `m.polH`, creating `m.pdbq`;
5. Checks charges in `m.pdbq`, all errors held in `m.err;`
6. Creates `s.gpf`, a parameter file for **AutoGrid**, based on ligand file `s.pdbq`;
7. Creates `s.vol`, a volume dimensions file; and finally,
8. Creates `s.dpf`, a parameter file for **AutoDock**, based on ligand file `s.pdbq`;

Its arguments are the stem of the filename of the macromolecule '`.pdb`' file and that of the ligand PDBQ file. See the flowchart below for more details. It shows what files are created by '`prepare`', and which scripts or programs are used. Steps 1.-4. are better carried out with a reliable molecular modeling system: these steps can produce some odd results unless carefully checked.

The user must check the `m.err` error file to ensure there are no non-integral charges, either on any residue in the macromolecule, or on the macromolecule as a whole. If there are, then the user must repair the `m.pdbq` file. This problem can arise if there are atoms for which no coordinates were assigned by the crystallographer, *e.g.* due to ambiguous electron density. Assuming there were no problems, `s.gpf` and `s.dpf` should be successfully produced.

## prepare-gpf+dpf

*Usage*: **`prepare-gpf+dpf macro lig`**

Executes only steps 6. through 8.

## rem-lp

*Usage*: **`rem-lp lig.pdbq`**
*or*: **`rem-lp lig.mol2`**
*Creates:* **`lig.pdbq`**
*or*: **`lig.mol2`**

This removes the lone-pairs (atom name = LP) added by some molecular modelling programs, such as SYBYL, and adds their partial charges on to that of the atom to which they were attached (SG in cysteines and SD in methionines). Otherwise, AutoDock treats lone-pairs as carbon atoms. (Note: if you need lone-pairs, you can force AutoGrid to calculate a grid map for “LP” atoms, using the atom code “L” in the “types” commands of AutoGrid and AutoDock).

## renumberatoms

*Usage*: **`lig.pdb > lig2.pdb`**

Used to renumber the atom IDs in the first column of the ATOM and HETATM records of a PDB file. Also updates the CONECT records appropriately.

## renumber-residues

*Usage*: **`lig.pdb > lig.rnm`**

Used by `prepare` to renumber residues in the macromolecule contiguously. This step is needed prior to using `protonate`, which may fail if there are gaps in the residue numbers.

## resrange

*Usage*: **`resrange lig.pdb`**

This is handy to summarise the range(s) of residues in a given protein PDB file.

## runtrj

*Usage*: **runtrj lig**
*Needs:* **lig.dpf**, **lig.trj**
*Creates:* **lig.tcom, lig.tlg** *and* **lig.tout**

This creates an **AutoDock** command file, `lig.tcom`, which is then used to convert the trajectory written in state variables (`lig.trj`), into a trajectory written in cartesian coordinates. `lig.trj` is created by an earlier run of **AutoDock**, in which `trjfrq` was set to a non-zero value.

## stats

*Usage*: **stats columns.dat**

This is a very useful, general `awk` program. Use it to calculate the minimum, maximum, mean and standard deviation for each column of numbers in an input file, here '`columns.dat`'. Any alphanumeric columns will be ignored.

# Appendix II: Parameters from AutoDock Version 1

### Table II.1: Lennard-Jones $C_6$ parameters (AutoDock version 1.0)

| | C | N | O | S | H |
|---|---|---|---|---|---|
| **C** | 1127.684 | 783.3452 | 633.7542 | 1476.364 | 226.9102 |
| **N** | 783.3452 | 546.7653 | 445.9175 | 1036.932 | 155.9833 |
| **O** | 633.7542 | 445.9175 | 368.6774 | 854.6872 | 124.0492 |
| **S** | 1476.364 | 1036.932 | 854.6872 | 1982.756 | 290.0756 |
| **H** | 226.9102 | 155.9833 | 124.0492 | 290.0756 | 46.73839 |

### Table II.2: Lennard-Jones $C_{12}$ parameters (AutoDock version 1.0)

| | C | N | O | S | H |
|---|---|---|---|---|---|
| **C** | 1272653. | 610155.1 | 588883.8 | 1569268. | 88604.24 |
| **N** | 610155.1 | 266862.2 | 249961.4 | 721128.6 | 39093.66 |
| **O** | 588883.8 | 249961.4 | 230584.4 | 675844.1 | 38919.64 |
| **S** | 1569268. | 721128.6 | 675844.1 | 1813147. | 126821.3 |
| **H** | 88604.24 | 39093.66 | 38919.64 | 126821.3 | 1908.578 |

### Table II.3: Hydrogen bonding *12-10* parameters (AutoDock version 1.0)

| Atoms i-j | $C_{12}$ | $C_{10}$ |
|---|---|---|
| O-H | 75570. | 23850. |
| N-H | 75570. | 23850. |
| S-H | 2657200. | 354290. |

# Appendix III: AutoDock File Formats

The formats will sometimes be given with notation such as ’%d’ to indicate a decimal integer; ’%6.3f’ for a floating point number with up to 6 characters and 3 digits after the decimal place; or ’%-7s’ for a left-justified string 7 characters wide. This notation is compatible with C, C++, awk/ nawk/gawk, and with a slight modification, Python.

The “◊” symbol is used to indicate one space.

The type of an argument is described using the following style:

`<string>` = an alphanumeric string. In most cases, this is a valid filename;
`<character>` = a single letter;
`<integer>` = a decimal integer;
`<positive_integer>` = a decimal integer greater than zero;
`<long_integer>` = a decimal integer in the “long” range (depends on computer);
`<float>` = a floating point or real number.

## III 1. Protein Data Bank with Partial Charges: PDBQ

***Extension:*** **`.pdbq`**

The name ’PDBQ’ derives from ’PDB’, the Protein DataBank, and ’Q’, a common symbol for partial charge. As the name suggests, the PDBQ format is very similar to the PDB format for ATOM records, with a modification in columns 71-76 (counting the first column as 1, not 0) to carry the partial charge, as %6.3f. Thus, the format of the whole line is as follows:

```
"ATOM◊◊%5d◊%-4s%1s%-3s◊%1s%4d%1s◊◊◊%8.3f%8.3f%8.3f%6.2f%6.2f%4s%6.3f\n",
atom_serial_num, atom_name, alt_loc, res_name, chain_id, res_num, ins_code, x, y, z,
occupancy, temp_factor, footnote, partial_charge
```

In addition to this, there are various records (ROOT, TORSION and BRANCH, for example) in the ligand PDBQ file that specify which portion of the molecule is rigid and which is flexible.

## III 2. PDBQ with Solvation Parameters: PDBQS

***Extension:*** **`.pdbqs`**

This format is derived from the PDBQ format, and is used to specify the atomic solvation parameters for the macromolecule, hence the “S”. The format of the lines is:

```
"ATOM◊◊%5d◊%-4s%1s%-3s◊%1s%4d%1s◊◊◊%8.3f%8.3f%8.3f%6.2f%6.2f%4s%6.3f%8.2f%8.2f\n",
atom_serial_num, atom_name, alt_loc, res_name, chain_id, res_num, ins_code, x, y, z,
occupancy, temp_factor, footnote, partial_charge,
atomic_fragmental_volume, atomic_solvation_parameter
```

The atomic fragmental volume and solvation parameters are derived from the method of Stouten *et al*.[24]

## III 3. AutoGrid Grid Parameter File: GPF

***Extension:*** **`.gpf`**

The input file is often referred to as a “grid parameter file” or “GPF” for short. The scripts described in the appendices give these files the extension “.gpf”. In the grid parameter file, the user must specify the following spatial attributes of the grid maps:

1. the center of the grid map;

2. the number of grid points in each of the *x*-, *y*- and *z*-directions; and

3. the separation or spacing of each grid point.

In addition, the pairwise-atomic interaction energy parameters must be specified. The following lines are required for each ligand atom type, *Y*:

4. the grid map filename for atom type *Y*;

5. seven lines containing the non-bonded parameters for each pairwise-atomic interaction, in the following order: *Y*-C, *Y*-N, *Y*-O, *Y*-S, *Y*-H, *Y*-*X*, (*X* is any other atom type) and *Y*-*M* (*M* is a metal, say).

Using coefficients $C_n$, $C_m$, $n$ and $m$, the pairwise interaction energy, $V(r)$ is given by:

$$V(r) \approx \frac{C_n}{r^n} - \frac{C_m}{r^m}$$

Alternatively, the user can specify $r_{eqm}$, ε, $n$ and $m$:

$$V(r) \approx \frac{\frac{m}{n-m}\varepsilon r_{eqm}^{n}}{r^n} - \frac{\frac{n}{n-m}\varepsilon r_{eqm}^{m}}{r^m}$$

24. Stouten, P. F. W., Frömmel, C., Nakamura, H. and Sander, C. (1993). “An effective solvation term based on atomic occupancies for use in protein simulations”, *Molecular Simulations*, **10**, 97-120.

This latter method of specification is more intuitive for the user, while **AutoGrid** handles the calculation of the coefficients. By default, the *Y-X* and *Y-M* lines are copies of the *Y*-H line. But in some systems, such as receptors which consist of DNA/protein complexes, both sulphur *and* phosphorus can be present. In this scenario, the *Y-X* line can be used for modeling interactions with receptor-phosphorus atoms. A very rough approximation for phosphorus parameters is to borrow those of carbon.

The "elecmap" line in the grid parameter file is the filename of the electrostatic potential grid map. The following parameter, "dielectric", if negative, indicates that the distance-dependent dielectric function of Mehler and Solmajer[3] will be used. If positive, however, the value of that number will be used as a constant dielectric. For example, if the value were 40.0, then a constant dielectric of 40 would be used.

The **AutoGrid** parameter file format is described below.

### AutoGrid Keywords and Commands

**`receptor <string>`**

Macromolecule filename, in PDBQ format.

**`gridfld <string>`**

The grid field filename, which will be written in a format readable by **AutoDock** and AVS[25]. The filename extension *must* be '.fld'.

**`npts <integer> <integer> <integer>`**

Number of *x*-, *y*- and *z*-grid points. Each *must* be an even integer number. When added to the central grid point, there will be an odd number of points in each dimension. The number of *x*-, *y*- and *z*-grid points need not be equal.

**`spacing <float>`**

The grid point spacing, in Å (see the diagram on page 8). Grid points must be uniformly spaced in AutoDock: this value is used in each dimension.

**`gridcenter <float> <float> <float>`**
**`gridcenter auto`**

The user can explicitly define the center of the grid maps, respectively the *x*, *y* and *z* coordinates of the center of the grid maps (units: Å, Å, Å.) Or the keyword "auto" can be given, in which case **AutoGrid** will center the grid maps on the center of mass of the macromolecule.

**`types <string>`**

1-letter names of the atom types present in the ligand; *e.g.* if there are carbons, nitrogens, oxygens and hydrogens, then this line will be "`CNOH`"; there are no delimiters.

25. "AVS" stands for "Application Visualization System"; AVS is a trademark of Advanced Visual Systems Inc., 300 Fifth Avenue, Waltham, MA 02154.

Appendices

**smooth <float>**

This is always 0.5Å, when using the AutoDock 3.0 free energy function. It is used to smooth the pairwise atomic affinity potentials (both van der Waals and hydrogen bonds). See the Theory section for more details.

**map <string>**

Filename of the grid map, for ligand atom type Y; the extension is usually ".map".

**nbp_coeffs <float> <float> <integer> <integer>**

Either "nbp_coeffs" or "nbp_r_eps" keywords can be used to define Lennard-Jones or hydrogen bond interaction energy parameters. The keyword "nbp_coeffs" specifies coefficients and exponents, in the order "$C_n$ $C_m$ $n$ $m$", delimited by spaces; $n$ and $m$ are integer exponents. The units of $C_n$ and $C_m$ must be kcal $mol^{-1}$ $Å^n$ and kcal $mol^{-1}$ $Å^m$ respectively; $n$ and $m$ have no units.

**nbp_r_eps <float> <float> <integer> <integer>**

Alternatively, the user can employ "nbp_r_eps" to specify the equilibrium distance and well depth, epsilon, for the atom pair. The equilibrium separation has units of Å and the well depth, epsilon, units of kcal $mol^{-1}$. The integer exponents $n$ and $m$ must be specified too.

In either case, the order of the parameters must be: *Y*-C, *Y*-N, *Y*-O, *Y*-S, *Y*-H, *Y*-X, and *Y*-M. Repeat 1 "map" line and the 7 "nbp_coeffs"or "nbp_r_eps" lines, for each atom type, *Y*, present in the ligand being docked.

**sol_par <float> <float>**

This is used to define the atomic fragmental volume and solvation parameters, and should not be changed from the Stouten values used to calibrate the AutoDock 3.0 free energy function.

**constant <float>**

This is added to all the values in a grid map, and is only set to a non-zero, positive number for hydrogen bonding maps. This value is essentially the penalty for un-formed hydrogen bonds in the complex.

**elecmap <string>**

Filename for the electrostatic potential energy grid map to be created; filename extension '.map'.

**dielectric <float>**

Dielectric function flag: if negative, **AutoGrid** will use *distance-dependent* dielectric of Mehler and Solmajer[3]; if the float is positive, **AutoGrid** will use this value as the dielectric constant.

**fmap <string>**

*(Optional.)* Filename for the so-called "floating" grid map[26]; filename extension '.map'. In such

26. This grid map is not used in AutoDock 3.0; its utility is under investigation, and may be included in a later version.

Appendices

floating grids, the scalar at each grid point is the distance to the nearest atom in the receptor. These values could be used to guide the docking ligand towards the receptor's surface, thus avoiding non-interesting, empty regions.

## III 4. Grid Map File

***Extension:*** **`.map`**

The first six lines of each grid map hold header information which describe the spatial features of the maps and the files used or created. These headers are checked by **AutoDock** to ensure that they are appropriate for the requested docking. The remainder of the file contains grid point energies, written as floating point numbers, one per line. They are ordered according to the nested loops z( y( x ) ). A sample header from a grid map is shown below:

---

```
GRID_PARAMETER_FILE vac1.nbc.gpf
GRID_DATA_FILE 4phv.nbc_maps.fld
MACROMOLECULE 4phv.new.pdbq
SPACING 0.375
NELEMENTS 50 50 80
CENTER -0.026 4.353 -0.038
125.095596
123.634560
116.724602
108.233879
:
```

---

## III 5. Grid Map Field File

***Extension:*** **`.maps.fld`**

This is essentially two files in one. It is both an AVS field file, and and **AutoDock** input file with **AutoDock**-specific information 'hidden' from AVS in the comments at the head of the file. **AutoDock** uses this file to check that all the maps it reads in are compatible with one-another and itself. For example, in this file, the grid spacing is 0.375 Angstroms, there are 60 intervals in each dimension, the grid is centered near (46,44,14), it was calculated around the macromolecule '`2cpp.pdbqs`', and the **AutoGrid** parameter file used to create this and the maps was '`2cpp.gpf`'. This file also points to a second file, '`2cpp.maps.xyz`', which contains the minimum and maximum extents of the grid box in each dimension, *x, y,* and *z*. Finally, it lists the grid map files that were calculated by **AutoGrid**, here '`2cpp.C.map`', '`2cpp.O.map`' and '`2cpp.e.map`'.

---

```
# AVS field file
#
# AutoDock Atomic Affinity and Electrostatic Grids
#
# Created by autogrid3.
#
#SPACING 0.375
#NELEMENTS 60 60 60
#CENTER 46.508 44.528 14.647
#MACROMOLECULE 2cpp.pdbqs
#GRID_PARAMETER_FILE 2cpp.gpf
#
ndim=3                  # number of dimensions in the field
dim1=61                 # number of x-elements
dim2=61                 # number of y-elements
dim3=61                 # number of z-elements
nspace=3                # number of physical coordinates per point
veclen=3                # number of affinity values at each point
data=float              # data type (byte, integer, float, double)
field=uniform           # field type (uniform, rectilinear, irregular)
coord 1 file=2cpp.maps.xyz filetype=ascii offset=0
coord 2 file=2cpp.maps.xyz filetype=ascii offset=2
coord 3 file=2cpp.maps.xyz filetype=ascii offset=4
label=C-affinity        # component label for variable 1
label=O-affinity        # component label for variable 2
label=Electrostatics    # component label for variable 2
#
# location of affinity grid files and how to read them
#
variable 1 file=2cpp.C.map filetype=ascii skip=6
variable 2 file=2cpp.O.map filetype=ascii skip=6
variable 3 file=2cpp.e.map filetype=ascii skip=6
```

## III 6. AutoDock Docking Parameter File: DPF

***Extension:*** **`.dpf`**

**AutoDock 3.0** has an interface based on keywords. This is intended to make it easier for the user to set up and control a docking job, and for the programmer to add new commands and functionality. The input file is often referred to as a “docking parameter file” or “DPF” for short. The scripts described in the appendices give these files the extension “.dpf”.

All delimiters where needed are white spaces. Default values, where applicable, are given in square brackets [thus]. A comment must be prefixed by the “#” symbol, and can be placed at the end of a parameter line, or on a line of its own.

Although ideally it should be possible to give these keywords in any order, not every possible combination has been tested, so it would be wise to stick to the following order.

### Command to set the seed for the random number generator

```
seed <long_integer>
seed time
seed pid
seed <long_integer> <long_integer>
seed time <long_integer>
seed <long_integer> time
seed time pid
seed pid <long_integer>
seed <long_integer> pid
seed pid time
```

There are two possible random number generator libraries. One is the system's own implementations, and the second is the platform-independent library from the University of Texas Biomedical School. If the user gives just one argument to "`seed`", then AutoDock will use the system's implementation of the random number generator and corresponding system seed call. On most platforms, these are "drand48" and "srand48". The platform-independent library, however, requires two seed values. Giving two arguments to "seed" tells AutoDock to use the platform-independent library for random number generation.

The random-number generator (RNG) for each docking job can be 'seeded' with either a user-defined, a time-dependent, or process-ID-dependent seed. These two seeds can be any combination of explicit long integers, the keyword "`time`" or the keyword "`pid`". When two arguments to seed are given, the portable RNG is used; when one is given, the built-in RNG (usually the "drand48" C-function) is used. The portable RNG is required for the genetic algorithm and the Solis and Wets routines. The portable RNG cannot be used with the simulated annealing routine: this needs just one seed parameter. The keyword, "time" gives the number of seconds since the epoch. The epoch is referenced to 00:00:00 CUT (Coordinated Universal Time) 1 Jan 1970. The "pid" gives the UNIX process ID of the currently executing AutoDock process, which is reading this parameter file.

### Parameters defining the grid maps to be used

**`types <string>`**

Atom names for all atom types present in ligand. Each must be a single character, and only one of: C, N, O, S, H, X, or M. The maximum number of characters allowed in this line is ATOM_MAPS, which is defined in the "autodock.h" include file. Do not use any spaces to delimit the types: they are not needed.

**`fld <string>`**

Grid data field file created by **AutoGrid** and readable by **AVS** (must have the extension ".fld").

**`map <string>`**

Filename for the first **AutoGrid** affinity grid map of the 1st atom type. This keyword plus filename must be repeated for all atom types in the order specifed by the "`types`" command. In all map files a 6-line header is required, and energies must be ordered according to the nested loops z( y( x

) ).

**map <string>**

Filename for the electrostatics grid map. 6-line header required, and energies must be ordered according to the nested loops z( y( x ) ).

### *Parameters defining the ligand and its initial state*

**move <string>**

Filename for the ligand to be docked. This contains most importantly, atom names, xyz-coordinates, anb partial atomic charges in PDBQ format. (Filename extension should be ".pdbq").

**about <float> <float> <float>**

Use this keyword to specify the center of the ligand, *about* which rotations will be made. (The coordinate frame of reference is that of the ligand PDBQ file.) Usually the rotation center of the ligand is the mean *x,y,z*-coordinates of the molecule. Inside **AutoDock**, the "`about`" xyz-coordinates are *subtracted* from each atom's coordinates in the input PDBQ file. So internally, the ligand's coordinates become centered at the origin. Units: Å, Å, Å.

**tran0 <float> <float> <float>**
**tran0 random**

Initial coordinates for the center of the ligand, in the same frame of reference as the receptor's grid maps. The ligand, which has been internally centered using the "about" coordinates, has the xyz-coordinates of the initial translation "`tran0 x y z`" *added* on. Every run starts the ligand from this location.

Alternatively, the user can just give the keyword "random" and **AutoDock** will pick random initial coordinates instead.

If there are multiple runs defined in this file, using the keyword "runs", then each new run will begin at this same location.

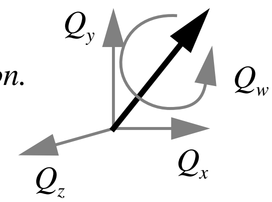
The user *must* specify the absolute *starting* coordinates for the ligand, used to start each run. The user should ensure that the ligand, when translated to these coordinates, still fits within the volume of the grid maps. If there are some atoms which lie outside the grid volume, then **AutoDock** will automatically correct this, until the ligand is pulled completely within the volume of the grids. (This is necessary in order to obtain complete information about the energy of the initial state of the system.) The user will be notified of any such changes to the initial translation by **AutoDock**. (Units: Å, Å, Å.)

**quat0 <float> <float> <float> <float>**
**quat0 random**

[1, 0, 0, 0°]
Respectively: $Q_x$, $Q_y$, $Q_z$, $Q_w$. Initial quaternion (applied to ligand) - $Q_x$, $Q_y$, $Q_z$ define the unit vector of the direction of rigid body rotation, and $Q_w$ defines the angle of rotation about this unit vector,

in ˚ . (Units: none,none,none, ˚.)

*A quaternion.*




Alternatively, the user can just give the keyword “random” and **AutoDock** will pick a random unit vector and a random rotation (between 0˚ and 360˚) about this unit vector. Each run will begin at this same random rigid body rotation.

**ndihe <integer>**

Number of dihedrals or rotatable bonds in the ligand. This may be specifed only if rotatable bonds have been defined using ROOT, BRANCH, TORS *etc*. keywords in the PDBQ file named on the “move” line. The number supplied to this command must agree with the number of torsions defined in this ligand PDBQ file. If this keyword is used, then the next keyword, **dihe0**, must also be specified. Note that if **ndihe** and **dihe0** are not specified and there are defined torsions in the ligand PDBQ file, **AutoDock** assumes that the $chi_1$, $chi_2$, $chi_3$, *etc.* are all zero, and does not change the initial ligand torsion angles. (See also “torsdof” below).

**dihe0 <float> ...**

Initial **relative** dihedral angles; there must be **ndihe** floating point numbers specified on this line. Each value specified here will be added to the corresponding torsion angle in the input PDBQ file, at the start of each run. Torsion angles are only specified by two atoms, so the definition of rotations is relative. Units: ˚.

### *Parameters defining ligand step sizes*

**tstep <float>**
**tstep <float> <float>**

[2.0 Å]
The first form, with one argument, defines the maximum translation jump for the first cycle that the ligand may make in one simulated annealing step. When “trnrf” is less than 1, the reduction factor is multiplied with the tstep at the end of each cycle, to give the new value for the next cycle. The second form allows the user to specify the value for the first cycle and the last cycle: AutoDock then calculates the reduction factor that satisfies these constraints. Units: Å.

**qstep <float>**

[50.0˚]
Maximum orientation step size for the angular component, *w*, of quaternion. Units: ˚.

**dstep <float>**

[50.0˚]

Maximum dihedral (torsion) step size. Units: ˚.

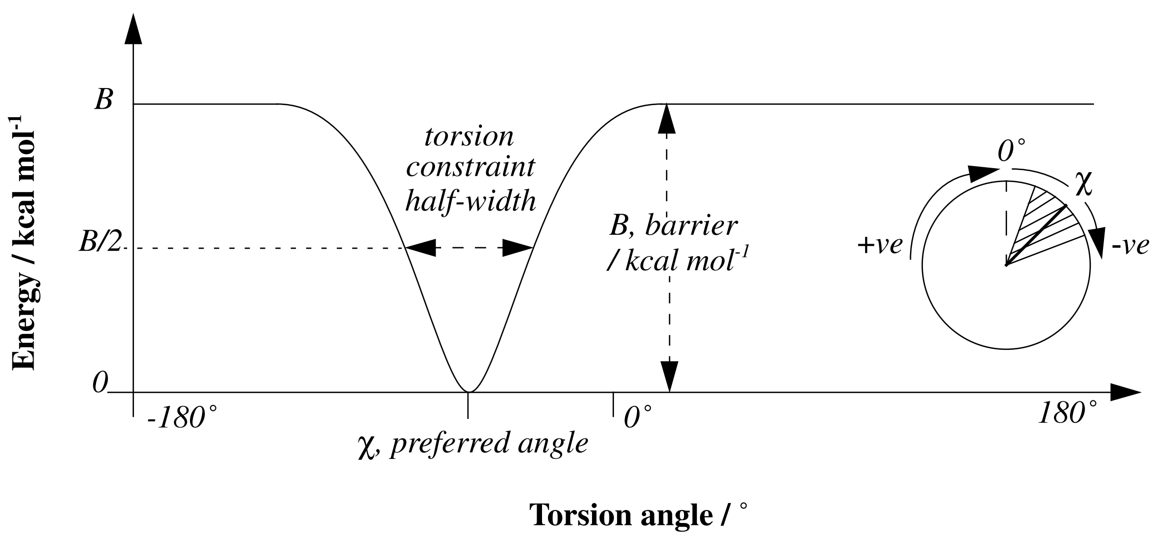
### Parameters defining optional ligand torsion constraints

**`barrier <float>`**

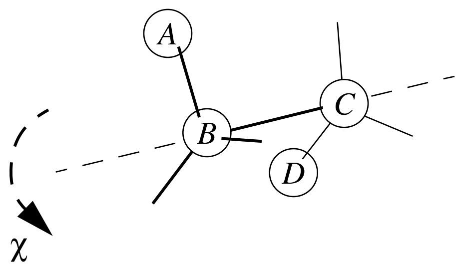
[10000.0]
*(Optional)* This defines the energy-barrier height applied to constrained torsions. When the torsion is at a preferred angle, there is no torsion penalty: this torsion's energy is zero. If the torsion angle falls within a disallowed zone, however, it can contribute up to the full barrier energy. Since the torsion-energy profiles are stored internally as arrays of type 'unsigned short', only positive integers between 0 and 65535 are allowed.

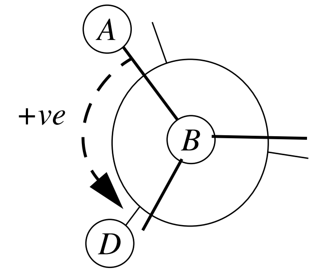
**`gausstorcon <integer> <float> <float>`**

*(Optional)* Adds a constraint to a torsion. The torsion number is identified by an integer. This identifier comes from the list at the top of the AutoTors-generated input ligand PDBQ file (on the REMARK lines). An energy profile will be calculated for this torsion. An inverted Gaussian bell curve is added for each new constraint. To completely specify each Gaussian, two floating point numbers are needed: the *preferred angle* and the *half-width* respectively (both in degrees). Note that the preferred angle should be specified in the range -180˚ to +180˚; numbers outside this range will be wrapped back into this range. This angle, $\chi$, is ***relative*** to the original torsion angle in the input structure. The *half-width* is the difference between the two angles at which the energy is half the barrier (B/2 in the diagram above). The smaller the half-width, the tighter the constraint.




If you wish to constrain to absolute-valued torsion angles, it will be necessary to zero the initial torsion angles in the ligand, before input to **AutoTors**. The problem arises from the ambiguous 2-atom definition of the rotatable bond *B-C*. To identify a torsion angle unambiguously, 4 atoms must be specified: *A*-*B*-*C*-*D*:




*looking down the B-C bond*

The sign convention for torsion angles which we use is anti-clockwise (counter-clockwise) are positive angles, clockwise negative. In the above diagram, looking down the bond *B*-*C*, the dihedral angle *A*-*B*-*C*-*D* would be positive.

There is no limit to the number of constraints that can be added to a given torsion. Each new torsion-constraint energy profile is combined with the pre-existing one by selecting the minimum energy of either the new or the existing profiles.

**`showtorpen`**

*(Optional) (Use only with "gausstorcon")* This switches on the storage and subsequent output of torsion energies. During each energy evaluation, the penalty energy for each constrained torsion, as specified by the "gausstorcon" command, will be stored in an array. At the end of each run, the final docked conformation's state variables are output, but with this command, the penalty energy for each torsion will be printed alongside its torsion angle.

**`hardtorcon <integer> <float> <float>`**

*(Optional)* This command also adds a torsion constraint to the <integer>-th torsion, as numbered in the AutoTors-generated REMARKs. The first float defines the *preferred relative angle*, and the second specifies the *full width* of the allowed range of torsion angles (both in degrees). This type of torsion constraint is "hard" because the torsion is never allowed to take values beyond the range defined. For example, "hardtorcon 3 60. 10." would constrain the third torsion to values between 55˚ and 65˚.

### *Parameter affecting torsional free energy*

**`torsdof <integer> <float>`**

[0, 0.3113]
This specifies respectively the number and the coefficient of the torsional degrees of freedom (DOF) for the estimation of the change in free energy upon binding, $\Delta G_{binding}$. For the purposes of AutoDock 3.0, the number of torsional DOF is the number of rotatable bonds in the ligand, excluding any torsions that rotate one or more hydrogen atoms, *e.g.* hydroxyls, amines, methyls. By default, the coefficient is 0.3113 kcal $mol^{-1}$, although the user can override this as necessary. (Units: none; kcal $mol^{-1}$).

Appendices

### Parameters for ligand internal energies

**`intnbp_coeffs <float> <float> <integer> <integer>`**

Respectively: $C_n$; $C_m$; $n$; $m$. This command specifies the internal pairwise non-bonded energy parameters for flexible ligands, where:

$$V(r) \approx \frac{C_n}{r^n} - \frac{C_m}{r^m}$$

These parameters are needed even if no rotatable bonds were defined in the ligand-PDBQ file. They are only used in the internal energy calculations for the ligand and must be consistent with those used in calculating the grid maps. (Units: kcal $mol^{-1}$ Å$^n$; kcal $mol^{-1}$ Å$^m$; none; none, respectively).

**`intnbp_r_eps <float> <float> <integer> <integer>`**

Respectively: $r_{eqm}$; $\varepsilon$; $n$; $m$, This command is an alternative way of specifying the internal pairwise non-bonded energy parameters for flexible ligands, where **AutoDock** calculates the pairwise atomic potential using:

$$V(r) \approx \frac{\frac{m}{n-m}\varepsilon r_{eqm}^{n}}{r^n} - \frac{\frac{n}{n-m}\varepsilon r_{eqm}^{m}}{r^m}$$

The first two arguments specify the equilibrium distance and well depth, epsilon, for the atom pair. The equilibrium separation has units of Å and the well depth, epsilon, units of kcal $mol^{-1}$. The integer exponents $n$ and $m$ must be specified too. Obviously, $n \neq m$. (Units: Å; kcal $mol^{-1}$; none; none, respectively).

**`intelec`**

*(Optional)* Internal ligand electrostatic energies will be calculated; the products of the partial charges in each non-bonded atom pair are pre-calculated, and output. Note that this is only relevant for flexible ligands.

### Parameters for simulated annealing searches

**`rt0 <float>`**

[500. cal $mol^{-1}$].
Initial "annealing temperature"; this is actually the absolute temperature multiplied by the gas constant $R$. $R$ = 8.314 J $mol^{-1}$ $K^{-1}$ = 1.987 cal $mol^{-1}$ $K^{-1}$. (Units: cal $mol^{-1}$.)

**`rtrf <float>`**

Annealing temperature reduction factor, $g$ [0.95 $cycle^{-1}$]. See the equation at the bottom of page 5. At the end of each cycle, the annealing temperature is multiplied by this factor, to give that of the next cycle. This must be positive but < 1 in order to cool the system. Gradual cooling is recommended, so as to avoid "*simulated quenching*", which tends to trap systems into local minima.

**`linear_schedule`**
**`schedule_linear`**
**`linsched`**
**`schedlin`**

These keywords are all synonymous, and instruct **AutoDock** to use a linear or *arithmetic* temperature reduction schedule during *Monte Carlo* simulated annealing. Unless this keyword is given, a *geometric* reduction schedule is used, according to the **`rtrf`** parameter just described. If the linear schedule is requested, then any **`rtrf`** parameters will be ignored. The first simulated annealing cycle is carried out at the annealing temperature **`rt0`**. At the end of each cycle, the temperature is reduced by (**`rt0/cycles`**). The advantage of the linear schedule is that the system samples evenly across the temperature axis, which is vital in entropic calculations. Geometric temperature reduction schedules on the other hand, under-sample high temperatures and over-sample low temperatures.

**`runs <integer>`**

[10]
Number of automated docking runs.

**`cycles <integer>`**

[50]
Number of temperature reduction cycles.

**`accs <integer>`**

[100]
Maximum number of accepted steps per cycle.

**`rejs <integer>`**

[100]
Maximum number of rejected steps per cycle.

**`select <character>`**

[m]
State selection flag. This character can be either **`m`** for the *minimum* state, or **`l`** for the *last* state found during each cycle, to begin the following cycle.

**`trnrf <float>`**

[1.0]
Per-cycle reduction factor for translations.

**`quarf <float>`**

[1.0]
Per-cycle reduction factor for quaternions.

**dihrf <float>**

Per-cycle reduction factor for dihedrals [1.].

### *Parameter to set the amount of output*

**outlev <integer>**

[1]
Diagnostic output level. For SA (simulated annealing): 0 = no output, 1 = minimal output, 2 = full state output at end of each cycle; 3 = detailed output for each step. For GA and GA-LS (genetic algorithm-local search): 0 = minimal output, 1 = write minimum, mean, and maximum of each state variable at the end of every generation. Use “outlev 1” for SA, and “outlev 0” for GA and GA-LS. If you use “outlev 1” with GA-LS, you will generate very large log files.

### *Parameters for trajectory output during SA dockings*

**trjfrq <integer>**

[0]
Output frequency, $n$, for trajectory of ligand, in steps. If $n = 0$, then no trajectory states will be output; otherwise, every $n^{th}$ state will be output. The *state* consists of 7 floats describing the $x,y,z$ translation, the $x,y,z$ components of the quaternion unit vector, the angle of rotation about the quaternion axis; and any remaining floats describing the torsions, in the same order as described in the input ligand PDBQ file).

**trjbeg <integer>**

[1]
Begin sampling states for trajectory output at the cycle with this value.

**trjend <integer>**

[50]
End trajectory output at this cycle.

**trjout <string>**

[lig.trj]
Trajectory filename. AutoDock will write out state variables to this file every “trjfrq” steps. Use the “traj” command in AutoDock’s command mode to convert this trajectory of state-variables into a series of PDB frames. The “traj” command is described in § “Using the Command Mode in AutoDock”; see also § “Trajectory Files”.

**trjsel <string>**

[E]
Trajectory output flag, can be either ‘**A**’ or ‘**E**’; the former outputs only *accepted* steps, while the latter outputs *either* accepted or rejected steps.

**watch <string>**

(*Optional)* Creates a "watch" file for real-time monitoring of an *in-progress* simulated annealing job. This works only if the "trjfrq" parameter is greater than zero. The watch file will be in PDB format, so give a ".pdb" extension. This file has an exclusive lock placed on it, while AutoDock is writing to it. Once the file is closed, the file is unlocked. This can signal to a watching visualization program that the file is complete and can now be read in, for updating the displayed coordinates. This file is written at exactly the same time as the trajectory file is updated

### Parameter for energies of atoms outside the grid

**`extnrg <float>`**

[1000.]
External grid energy assigned to any atoms that stray outside the volume of the grid during a docking. Units: kcal $mol^{-1}$.

### Parameter for initializing the ligand in SA

**`e0max <float> <positive_integer>`**

[0., 10000]
This is only used by the simulated annealing method. This keyword stipulates that the ligand's initial state cannot have an energy greater than the first value, nor can there be more than the second value's number of retries. Typical energy values range from 0 to 1000 kcal/mol. If the initial energy exceeds this value, a new random state is generated and tested. This process is iterated until the condition is satisfied. This can be particularly useful in preventing runs starting in exceptionally high energy regions. In such cases, the ligand can get trapped because it is unable to take a long enough translational jump. In those grids were the ligand is small enough to fit into the low energy regions with ease, there will not be many iterations before a favorable location is found. But in highly constrained grids, with large ligands, this initialization loop may run almost indefinitely.

### Parameters for cluster analysis of docked conformations

**`rmsref <string>`**

The root mean square deviation (rmsd) of the docked conformations will be calculated with respect to the coordinates in the PDB or PDBQ file specified here. This is useful when the experimentally determined complex conformation of the ligand is known. The order of the atoms in this file must match that in the input PDBQ file given by the **`move`** command. These values of rmsd will be output in the last column of the final PDBQ records, after the clustering has been performed.

**`rmstol <float>`**

[0.5Å]
rms deviation tolerance for cluster analysis or 'structure binning' , carried out after multiple docking runs. If two conformations have an rms less than this tolerance, they will be placed in the same cluster. The structures are ranked by energy, as are the clusters. The lowest energy representative

from each cluster is output in PDBQ format to the log file. To keep the ligand's residue number in the input PDBQ file, use the '**-k**' flag; otherwise the clustered conformations are numbered incrementally from 1. (Units: Å).

**rmsnosym**

When more than one run is carried out in a given job, cluster analysis or 'structure binning' will be performed, based on structural rms difference, ranking the resulting families of docked conformations in order of increasing energy. The default method for structure binning allows for atom similarity, as in a tertiary-butyl which can be rotated by +/-120°, but in other cases it may be desirable to bypass this similar atom type checking and calculate the rms on a one-for-one basis. The symmetry checking algorithm scans all atoms in the reference structure, and selects the nearest atom of identical atom type to be added to the sum of squares of distances. This works well when the two conformations are very similar, but this assumption breaks down when the two conformations are translated significantly. Symmetry checking can be turned off using the **rmsnosym** command; omit this command if you still want symmetry checking.

### *Parameters for re-clustering the results of several jobs*

**cluster <string>**

*(Clustering multi-job output only.)* **AutoDock** will go into 'cluster mode'. Use this command only to perform cluster analysis on the combined output, *<PDBQfilename>*, of several jobs. This command can be very useful when many jobs have been distributed to several machines and run in 'parallel'. The docking parameter file will need the following keywords: **rmstol** and **types**; and optionally write_all_cluster_members and/or **rmsnosym**.

It is necessary to `grep` the USER lines along with the ATOM records, since **AutoDock** parses the these lines to determine what the energy of that particular conformation was. For more information, see the example DPF files given later.

**write_all_cluster_members**

*(Clustering multi-job output only.)* This command is used only with the **cluster** command, to write out all members of each cluster instead of just the lowest energy from each cluster. This affects the cluster analysis PDBQ output at the end of each job.

### *Parameters for genetic algorithm, Lamarckian GA and evolutionary programming searches*

**ga_pop_size <positive_integer>**

[50]
This is the number of individuals in the population. Each individual is a coupling of a genotype and its associated phenotype. Usually, this number is fixed throughout the run. Typical values range from 50 to 200.

**ga_num_evals <positive_integer>**

[250000]

This is the maximum number of energy evaluations that a GA run should make.

**ga_num_generations <positive_integer>**

[27000]
This is the maximum number of generations that a GA or LGA run should last.

**ga_elitism <integer>**

[1]
This is used in the selection mechanism of the GA. This is the number of top individuals that are guaranteed to survive into the next generation.

**ga_mutation_rate <float>**

[0.02]
This is a floating point number from 0 to 1, representing the probability that a particular gene is mutated. This parameter is typically small.

**ga_crossover_rate <float>**

[0.80]
This is a floating point number from 0 to 1 denoting the crossover rate. Crossover rate is the expected number of pairs in the population that will exchange genetic material. Setting this value to 0 turns the GA into the evolutionary programming (EP) method, but EP would probably require a concomitant increase in the ga_mutation_rate in order to be effective.

**ga_window_size <positive_integer>**

[10]
This is the number of preceding generations to take into consideration when deciding the threshold for the worst individual in the current population.

**ga_cauchy_alpha <float>**

[0]

**ga_cauchy_beta <float>**

[1]
These are floating point parameters used in the mutation of real number genes. They correspond to the alpha and beta parameters in a Cauchy distribution. Alpha roughly corresponds to the mean, and beta to something like the variance of the distribution. It should be noted, though, that the Cauchy distribution doesn't have finite variance. For the mutation of a real valued gene, a Cauchy deviate is generated and then added to the original value.

### *Command to set genetic algorithm parameters*

**set_ga**

This command sets the global optimizer to be a genetic algorithm [GA]. This is required to perform a GA search. This passes any '`ga_`' parameters specified **before** this line to the global optimizer object. If this command is omitted, or it is given before the 'ga_' parameters, your choices will not take effect, and the default values for the optimizer will be used.

To use the traditional genetic algorithm, do not specify the local search parameters, and do not use the "set_sw1" or "set_psw1" commands.

To use the **Lamarckian genetic algorithm**, you **must** also specify the parameters for local search, and then issue either the 'set_sw1' or 'set_psw1' command. The former command uses the strict Solis and Wets local search algorithm, while the latter uses the pseudo-Solis and Wets algorithm: see earlier for details about how they differ.

### *Parameters for local search*

**sw_max_its <positive_integer>**

[50]
This is the maximum number of iterations that the local search procedure apply to the phenotype of any given individual. This is an unsigned integer. In Bill's experiments, he used a combination of iterations and function evaluations. It seems to me, that a value around 30 should be fine.

**sw_max_succ <positive_integer>**

[4]
This is the number of successes in a row before a change is made to the rho parameter in Solis & Wets algorithms. This is an unsigned integer and is typically around four.

**sw_max_fail <positive_integer>**

[4]
This is the number of failures in a row before Solis & Wets algorithms adjust rho. This is an unsigned integer and is usually around four.

**sw_rho <float>**

[1.0]
This is a parameter of the Solis & Wets algorithms. It defines the initial variance, and specifies the size of the local space to sample.

**sw_lb_rho <float>**

[0.01]
This is the lower bound on rho, the variance for making changes to genes (*i.e.* translations, orientation and torsions). rho can never be modified to a value smaller than "sw_lb_rho".

**ls_search_freq <float>**

[0.06]

This is the probability of any particular phenotype being subjected to local search.

### Commands to choose and set the local search method

Both of these commands, 'set_sw1' and 'set_psw1', pass any 'sw_' parameters set before this line to the local searcher. If you forget to use this command, or give it before the 'sw_' keywords, your choices will not take effect, and the default values for the optimizer will be used.

**set_sw1**

Instructs AutoDock to use the classical Solis and Wets local searcher, using the method of uniform variances for changes in translations, orientations and torsions.

**set_psw1**

Instructs AutoDock to use the pseudo-Solis and Wets local searcher. This method maintains the relative proportions of variances for the translations in Å and the rotations in radians. These are typically 0.2 Å and 0.087 radians to start with, so the variance for translations will always be about 2.3 times larger than that for the rotations (*i.e.* orientation and torsions).

### Commands to perform automated docking

**simanneal**

This command instructs AutoDock to do the specifed number of docking runs using the simulated annealing (SA) search engine. This uses the value set by the "runs" keyword as the number of SA docking runs to carry out. All relevant parameters for the simulated annealing job must be set first. These are indicated above by **[SA]** in each keyword description.

**do_local_only <integer>**

This keyword instructs AutoDock to carry out only the local search of a global-local search; the genetic algorithm parameters are ignored, with the exception of the population size. This is an ideal way of carrying out a minimization using the same force field as is used during the dockings. The "ga_run" keyword should not be given. The number after the keyword determines how many dockings will be performed.

**do_global_only <integer>**

This keyword instructs AutoDock to carry out dockings using only a global search, i.e. the traditional genetic algorithm. The local search parameters are ignored. The "ga_run" keyword should not be given. The number after the keyword determines how many dockings will be performed.

**ga_run <integer>**

[10]
This command invokes the new hybrid, Lamarckian genetic algorithm search engine, and performs the requested number of dockings. All appropriate parameters must be set first: these are listed above by "ga_".

### *Command to perform clustering of docked conformations*

**`analysis`**

This performs a cluster analysis on results of a docking, and outputs the results to the log file. The docked conformations are sorted in order of increasing energy, then compared by root mean square deviation. If the conformer is within the "rmstol" threshold, it is placed into the same cluster. A histogram is printed showing the number in each cluster, and if more than one member, the cluster's mean energy. Furthermore, a table is printed to the docking log file of cluster rmsd and reference rmsd values.

# Appendix IV: Example Parameter Files

## IV 1. AutoGrid GPF

An example **AutoGrid** parameter file is given below:

---

```
receptor 3ptb.pdbqs              #macromolecule
gridfld  3ptb.maps.fld           #grid_data_file
npts     60 60 60                #num.grid points in xyz
spacing  .375                           #spacing (Angstroms)
gridcenter -1.853 14.311 16.658#xyz-coordinates or 'auto"
types CANH                              #atom type names
smooth 0.500                            #store minimum energy within radius (Angstroms)
map 3ptb.C.map                          #filename of grid map
nbp_r_eps  4.00 0.0222750 12  6#C-C lj
nbp_r_eps  3.75 0.0230026 12  6#C-N lj
nbp_r_eps  3.60 0.0257202 12  6#C-O lj
nbp_r_eps  4.00 0.0257202 12  6#C-S lj
nbp_r_eps  3.00 0.0081378 12  6#C-H lj
nbp_r_eps  3.00 0.0081378 12  6#C-H lj
nbp_r_eps  3.00 0.0081378 12  6#C-H lj
sol_par 12.77 0.6844             #C atomic fragmental volume, solvation param.
constant 0.000                          #C grid map constant energy
map 3ptb.A.map                          #filename of grid map
nbp_r_eps  4.00 0.0222750 12  6#A-C lj
nbp_r_eps  3.75 0.0230026 12  6#A-N lj
nbp_r_eps  3.60 0.0257202 12  6#A-O lj
nbp_r_eps  4.00 0.0257202 12  6#A-S lj
nbp_r_eps  3.00 0.0081378 12  6#A-H lj
nbp_r_eps  3.00 0.0081378 12  6#A-H lj
nbp_r_eps  3.00 0.0081378 12  6#A-H lj
sol_par 10.80 0.1027             #A atomic fragmental volume, solvation param.
constant 0.000                          #A grid map constant energy
map 3ptb.N.map                          #filename of grid map
nbp_r_eps  3.75 0.0230026 12  6#N-C lj
nbp_r_eps  3.50 0.0237600 12  6#N-N lj
nbp_r_eps  3.35 0.0265667 12  6#N-O lj
nbp_r_eps  3.75 0.0265667 12  6#N-S lj
nbp_r_eps  2.75 0.0084051 12  6#N-H lj
nbp_r_eps  2.75 0.0084051 12  6#N-H lj
nbp_r_eps  2.75 0.0084051 12  6#N-H lj
sol_par  0.00 0.0000             #N atomic fragmental volume, solvation param.
constant 0.000                          #N grid map constant energy
map 3ptb.H.map                          #filename of grid map
nbp_r_eps  3.00 0.0081378 12  6#H-C lj
nbp_r_eps  2.75 0.0084051 12  6#H-N lj
nbp_r_eps  1.90 0.3280000 12 10#H-O hb
nbp_r_eps  2.50 0.0656000 12 10#H-S hb
nbp_r_eps  2.00 0.0029700 12  6#H-H lj
nbp_r_eps  2.00 0.0029700 12  6#H-H lj
nbp_r_eps  2.00 0.0029700 12  6#H-H lj
```

```
sol_par  0.00 0.0000              #H atomic fragmental volume, solvation param.
constant 0.118                           #H grid map constant energy
elecmap 3ptb.e.map                #electrostatic potential map
dielectric -0.1146                #<0,distance-dep.diel; >0,constant
#fmap 3ptb.f.map                  #floating grid
```

---

Note how hydrogen bonding is defined for oxygens. If a line in the parameter file contains a '10' in the fourth column, **AutoGrid** will treat this atom-pair as hydrogen bonding. So in the example above, the last 3 lines in the "mcp2_O.map" block will be treated as hydrogen bonds. **AutoGrid** scans for any polar hydrogens in the macromolecule. The vector from the hydrogen-donor, along with the vector from the probe-atom at the current grid point, are used to calculate the directional attenuation of the hydrogen bond. In this example, **AutoGrid** will calculate H-bonds between O-H, O-*X* and O-*M*.

## IV 2. AutoDock DPF

Some examples of commented **AutoDock** parameter files are given below.

### *Example 1: Docking using Monte Carlo Simulated Annealing*

In this case, the ligand file '`xk263pm3.pdbq`' has been defined such that it contains 10 rotatable bonds. The docking will be sampled every 7500 steps, from cycle 45 to cycle 50. Either accepted or rejected states will be output. The trajectory file '`xk263pm3.trj`' will hold the state information required to generate the coordinates later on. The external grid energy is set to 0.0, which can allow greater freedom for ligand rotations during docking.

---

```
seed random
types CNOH                        # atom type names

fld 4phv.nbc_maps.fld             # grid data file
map 4phv.nbc_C.map                # C-atomic affinity map
map 4phv.nbc_N.map                # N-atomic affinity map
map 4phv.nbc_O.map                # O-atomic affinity map
map 4phv.nbc_H.map                # H-atomic affinity map
map 4phv.nbc_e.map                # electrostatics map

move xk263pm3.pdbq                # ligand
about -5.452 -8.626 -0.082        # ligand center

tran0 -5.452 -8.626 -0.082        # initial coordinates/A
quat0 1. 0. 0. 0.              # initial quaternion:unit-vector(qx,qy,qz);angle/deg(qw)
ndihe 10                          # number of rotatable bonds
dihe0 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. # initial dihedrals/deg

tstep 0.2                         # translation step/A
qstep 5.                          # quaternion step/deg
```

Appendices

```
dstep 5.                        # torsion step/deg
trnrf 1.                        # trans reduction factor/per cycle
quarf 1.                        # quat reduction factor/per cycle
dihrf 1.                        # tors reduction factor/per cycle

intnbp_coeffs 1272653.000  1127.684  12   6 # C-C internal energy non-bond parameters
intnbp_coeffs  610155.100   783.345  12   6 # C-N internal energy non-bond parameters
intnbp_coeffs  588883.800   633.754  12   6 # C-O internal energy non-bond parameters
intnbp_coeffs   88604.240   226.910  12   6 # C-H internal energy non-bond parameters
intnbp_coeffs  266862.200   546.765  12   6 # N-N internal energy non-bond parameters
intnbp_coeffs  249961.400   445.918  12   6 # N-O internal energy non-bond parameters
intnbp_coeffs   39093.660   155.983  12   6 # N-H internal energy non-bond parameters
intnbp_coeffs  230584.400   368.677  12   6 # O-O internal energy non-bond parameters
intnbp_coeffs   38919.640   124.049  12   6 # O-H internal energy non-bond parameters
intnbp_coeffs    1908.578    46.738  12   6 # H-H internal energy non-bond parameters

rt0  500.                       # initial RT
rtrf 0.95                       # RT reduction factor/per cycle

runs 10                         # number of runs
cycles 50                       # cycles
accs   100                      # steps accepted
rejs   100                      # steps rejected
select m                        # minimum or last

outlev 1                        # diagnostic output level

rmstol 0.5                      # cluster tolerance/A

trjfrq 7500                     # trajectory frequency
trjbeg 45                       # start trj output at cycle
trjend 50                       # end trj output at cycle
trjout xk263pm3.trj             # trajectory file
trjsel E                        # A=acc only;E=either acc or rej

extnrg 0.0                      # external grid energy
e0max 0.0                       # maximum allowable energy to start a run

simanneal                       # perform automated docking using simulated annealing

analysis                        # perform a ranked cluster analysis
```

### *Example 2: Clustering Many Dockings*

The next example `DPF` shows how to use the cluster mode in **AutoDock**. The PDBQ files containing the final docked conformations have been extracted from the **AutoDock** log files (using the UNIX `grep` command), and stored together in the file “vac1.new.dlg.pdbq”. You can extract the “DOCKED:” records during the dockings, or after the dockings have finished. For example:

```
% egrep '^DOCKED: ' vac1.*.dlg | sed 's/^DOCKED: //' > vac1.grouped.dlg.pdbq
```

or:

```
% egrep '^ATOM |^HETATM|^REMARK|^USER  ' vac1.*.dlg > vac1.grouped.dlg.pdbq
```

The tolerance for the positional rms deviation is set to 1.5Å, so only conformations with this rms deviation or less will be placed in the same cluster. All conformations will be written out, instead of just the lowest energy representative from each conformationally distinct cluster.

You may include the `rmsnosym` command, if you do not wish to use symmetry checking while clustering. Also, you must finish the DPF with the `analysis` command, to instruct AutoDock to perform the clustering and write out the histogram of docked conformations.

```
types CANOH                          # atom_type_names
rmstol 1.5                           # cluster_tolerance/A
write_all                            # write all conformations in a cluster
cluster vac1.new.dlg.pdbq            # structure binning
analysis                             # do cluster analysis on results
```

### Example 3: Docking Using the Lamarckian Genetic Algorithm (LGA)

This DPF shows how to set up a docking using the genetic algorithm (GA) in combination with the pseudo-Solis and Wets local search algorithm (psw1). This is also known as the Lamarckian Genetic Algorithm or LGA.

```
seed    time pid                     # for random number generator
types   CANH                         # atom type names
fld     3ptb.maps.fld                # grid data file
map     3ptb.C.map                   # C-atomic affinity map
map     3ptb.A.map                   # A-atomic affinity map
map     3ptb.N.map                   # N-atomic affinity map
map     3ptb.H.map                   # H-atomic affinity map
map     3ptb.e.map                   # electrostatics map

move    benA.pdbq                    # small molecule
about   -1.853 14.311 16.658         # small molecule center
tran0   random                       # initial coordinates/A or "random"
quat0   random                       # initial quaternion or "random"
ndihe   0                            # number of initial torsions
dihe0   random                       # initial torsions
torsdof 0 0.3113                     # num. non-H tors.degrees of freedom & coeff.
tstep   0.2                          # translation step/A
qstep   5.                           # quaternion step/deg
dstep   5.                           # torsion step/deg
trnrf   1.                           # trans reduction factor/per cycle
quarf   1.                           # quat reduction factor/per cycle
dihrf   1.                           # tors reduction factor/per cycle
intnbp_r_eps  4.00 0.0222750  12  6     #C-C lj
intnbp_r_eps  4.00 0.0222750  12  6     #C-A lj
intnbp_r_eps  3.75 0.0230026  12  6     #C-N lj
intnbp_r_eps  3.00 0.0081378  12  6     #C-H lj
intnbp_r_eps  4.00 0.0222750  12  6     #A-A lj
```

```
intnbp_r_eps  3.75 0.0230026  12  6     #A-N lj
intnbp_r_eps  3.00 0.0081378  12  6     #A-H lj
intnbp_r_eps  3.50 0.0237600  12  6     #N-N lj
intnbp_r_eps  2.75 0.0084051  12  6     #N-H lj
intnbp_r_eps  2.00 0.0029700  12  6     #H-H lj
outlev  1                         # diagnostic output level
rmstol  0.5                       # cluster tolerance/A
rmsref  benA.pdbq                 # reference structure for RMS calc.
write_all                         # write all conformations in a cluster
extnrg  1000.                     # external grid energy
e0max   0. 10000                  # max. allowable initial energy, max. num. retries

ga_pop_size 50                    # number of individuals in population
ga_num_evals 150000               # maximum number of energy evaluations
ga_num_generations 27000          # maximum number of generations
ga_elitism 1                      # num. of top individuals that automatically survive
ga_mutation_rate 0.02             # rate of gene mutation
ga_crossover_rate 0.80            # rate of crossover
ga_window_size 10                 # num. of generations for picking worst individual
ga_cauchy_alpha 0                 # ~mean of Cauchy distribution for gene mutation
ga_cauchy_beta 1                  # ~variance of Cauchy distribution for gene mutation
set_ga                            # set the above parameters for GA

sw_max_its 300                    # number of iterations of Solis & Wets local search
sw_max_succ 4                     # number of consecutive successes before changing rho
sw_max_fail 4                     # number of consecutive failures before changing rho
sw_rho 1.0                        # size of local search space to sample
sw_lb_rho 0.01                    # lower bound on rho
ls_search_freq 0.06               # probability of performing local search on an indiv.
set_psw1                          # set the above pseudo-Solis & Wets parameters

ga_run 10                         # do this many hybrid GA-LS runs
analysis                          # do cluster analysis on results
```

# Appendix V: AutoDock References

## V 1. Primary References

### *AutoDock 3.0*

Morris, G. M., Goodsell, D. S., Halliday, R.S., Huey, R., Hart, W. E., Belew, R. K. and Olson, A. J. (1998), *J. Computational Chemistry*, **19**: 1639-1662. "Automated Docking Using a Lamarckian Genetic Algorithm and and Empirical Binding Free Energy Function".

*ABSTRACT: A novel and robust automated docking method that predicts the bound conformations of flexible ligands to macromolecular targets has been developed and tested, in combination with a new scoring function that estimates the free energy change upon binding. Interestingly, this method applies a Lamarckian model of genetics, in which environmental adaptations of an individual's phenotype are reverse transcribed into its genotype and become heritable traits (sic). We consider three search methods, Monte Carlo simulated annealing, a traditional genetic algorithm, and the Lamarckian genetic algorithm, and compare their performance in dockings of seven protein-ligand test systems having known three dimensional structure. We show that both the traditional and Lamarckian genetic algorithms can handle ligands with more degrees of freedom than the simulated annealing method used in earlier versions of AutoDock, and that the Lamarckian genetic algorithm is the most efficient, most reliable and most successful of the three. The empirical free energy function was calibrated using a set of 30 structurally-known protein-ligand complexes with experimentally-determined binding constants. Linear regression analysis of the observed binding constants in terms of a wide variety of structure-derived molecular properties was performed. The final model had a residual standard error of 9.11 kJ mol-1 (2.177 kcal mol-1) and was chosen as the new energy function. The new search methods and empirical free energy function are available in AutoDock version 3.0.*

### *AutoDock 2.4*

Morris, G. M., Goodsell, D. S., Huey, R. and Olson, A. J. (1996), *J. Computer-Aided Molecular Design*, **10**: 293-304. "Distributed automated docking of flexible ligands to proteins: Parallel applications of AutoDock 2.4".

*ABSTRACT: AutoDock 2.4 predicts the bound conformations of a small, flexible ligand to a nonflexible macromolecular target of known structure. The technique combines simulated annealing for conformation searching with a rapid grid-based method of energy evaluation based on the AMBER force field. AutoDock has been optimized in performance without sacrificing accuracy; it incorporates many enhancements and additions, including an intuitive interface. We have developed a set of tools for launching and analyzing many independent docking jobs in parallel on a heterogeneous network of UNIX-based workstations. This paper describes the current release, and the results of a suite of diverse test systems. We also present the results of a systematic investigation into the effects of varying simulated-annealing parameters on molecular docking. We show that even for ligands with a large number of degrees of freedom, root-mean-square deviations of less than 1 A from the crystallographic conformation are obtained for the lowest-energy dockings, although fewer dockings find the crystallographic conformation when there are more degrees of freedom.*

### AutoDock 1.0

Goodsell, D. S. and Olson, A. J. (1990), *Proteins: Str. Func. and Genet.*, **8**: 195-202. "Automated Docking of Substrates to Proteins by Simulated Annealing".

## V 2. Reviews of Applications

Goodsell, D. S., Morris, G. M. and Olson, A. J. (1996), *J. Mol. Recognition,* **9**: 1-5. "Docking of Flexible Ligands: Applications of AutoDock".

## V 3. Selected Applications and Citations of AutoDock

Minke, W.E., Diller, D.J., Hol, W.G., and Verlinde C. L. (1999), *J. Med. Chem.*, **42**: 1778-1788. "The role of waters in docking strategies with incremental flexibility for carbohydrate derivatives: heat-labile enterotoxin, a multivalent test case".

Laederach, A., Dowd, M.K., Coutinho, P.M., and Reilly, P.J. (1999), *Proteins:Structure, Function and Genetics*, **37**:166-175. "Automated Docking of Maltose, 2-Deoxymaltose, and Maltotetraose into the Soybean beta-Amylase Active Site".

Matias, P. M., Saraiva, L. M., Soares, C. M., Coelho, A. V., LeGall, J., and Armenia Carrondo, M. (1999) *JBIC*, **4**: 478-494. "Nine-haem cytochrome c from Desulfovibrio desulfuricans ATCC 27774: primary sequence determination, crystallographic refinement at 1.8 Å and modelling studies of its interaction with the tetrahaem cytochrome c3".

Bitomsky, W. and Wade, R. C. (1999), *J. Am. Chem. Soc.*, **121**: 3004-3013. "Docking of Glycosaminoglycans to Heparin-Binding Proteins: Validation for aFGF, bFGF, and Antithrombin and Application to IL-8".

Lorber, D. M. (1999), *Chemistry & Biology*, **6**: R227-R228. "Computational drug design".

Rao, M. S. and Olson, A. J. (1999), *Proteins:Structure, Function and Genetics*, **34**: 173-83. "Modelling of factor Xa-inhibitor complexes: a computational flexible docking approach".

Heine, A., Stura, E.A., Yli-Kauhaluoma, J.T., Gao, C., Deng, Q., Beno, B.R., Houk, K.N., Janda, K.D., and Wilson, I.A. (1998), *Science*, **279**: 1934-1940. "An antibody *exo* Diels-Alderase inhibitor complex at 1.95 Å resolution".

Coutinho, P. M., Dowd, M. K., Reilly, P. J., (1998), *Industrial & Engineering Chemistry Research*, **37**: 2148-2157."Automated Docking of -(1,4)- and -(1,6)-Linked Glucosyl Trisaccharides in the Glucoamylase Active Site."

Lozano, J. J., Lopez-de-Brinas, E., Centeno, N.B., Guigo, R. and Sanz, F. (1997), *J. Com-*

*puter-Aided Molecular Design*, **11**: 395-408. "Three-dimensional modelling of human cytochrome P450 1A2 and its interaction with caffeine and MeIQ".

Mahmoudian, M. (1997), *J. Molecular Graphics & Modelling*, **15**:149-153. "The cannabinoid receptor: Computer-aided molecular modeling and docking of ligand".

Coutinho, P. M., Dowd, M. K. and Reilly, P. J. (1997), *Proteins: Str. Func. and Genet.*, **28**:162-173. "Automated Docking of Glucosyl Disaccharides in the Glucoamylase Active Site".

Coutinho, P. M., Dowd, M. K. and Reilly, P. J. (1997), *Proteins: Str. Func. and Genet.*, **27**:235-248. "Automated Docking of Monosaccharide Substrates and Analogues and Melthyl alpha-Acarviosinide in the Glucoamylase Active Site".

Neurath, A. R., Jiang, S., Strick, K. L., Li, Y.-Y., and Debnath, A. K. (1996), *Nature Medicine*, **2**:230-234. "Bovine beta-lactoglobulin modified by 3-hydroxyphthalic anhydride blocks the CD4 cell receptor for HIV".

Gamper AM, Winger RH, Liedl KR, Sotriffer CA, Varga JM, Kroemer RT, Rode BM. (1996), *J. Med. Chem.*, **39**, 3882-3888. "Comparative molecular field analysis of haptens docked to the multispecific antibody IgE".

Sotriffer, C. A., Liedl, K. R., Winger, R. H., Gamper, A. M., Kroemer, R. T., Linthicum, D. S., Rode, B.-M. and Varga, J. M. (1996) *Molecular Immunology*, **33**: 129-144. "Heteroligation of a mouse monoclonal IgE antibody (La2) with small molecules, analysed by computer-aided automated docking".

Zhang, T. and Koshland, D. E. (1995 ), *Protein Science*, **4**: 84-92. "Modeling substrate binding in Thermus thermophilus isopropylmalate dehydrogenase".

Kedishvili, N. Y., Bosron, W. F., Stone, C. L., Hurley, T.D., Peggs, C. F., Thomasson, H. R., Popov, K. M., Carr, L. G., Edenberg, H. J. and Li, T.-K. (1995) *J. Biol. Chem.*, **270**: 3625-3630. "Expression and kinetic characterization of recombinant human stomach alcohol dehydrogenase".

Stone, C. L., Hurley, T. D., Peggs, C. F., Kedishvili, N. Y., Davis, G. J., Thomasson, H. R., Li, T.-K. and Bosron, W. F. (1995) *Biochemistry*, **34**: 4008-4014. "Cimetidine inhibition of human gastric and liver alcohol dehydrogenase isoenzymes: identification of inhibitor complexes by kinetic and molecular modeling".

Tummino, P. J., Ferguson, D., Jacobs, C. M ., Tait, B., Hupe, L., Lunney, E. and Hupe, D. (1995) *Arch. Biochem. Biophys.*, **316**: 523-528. "Competitive inhibition of HIV-1 protease by biphenyl carboxylic acids".

Friedman, A. R., Roberts, V. A. and Tainer, J. A. (1994) *Proteins: Str. Func. and Genet.*, **20**: 15-24. "Predicting molecular interactions and inducible complementarity: fragment docking of Fab-peptide complexes".

Lunney, E. A., Hagen, S. E., Domagala, J. M., Humblet, C., Kosinski, J., Tait, B. D., Warmus, J. S., Wilson, M., F erguson, D., Hupe, D., Tummino, P. J., Baldwin, E. T., Bhat, T. N., Liu, B. and Erickson, J. W. (1994) *J. Med. Chem.*, **37** : 2664-2677. "A novel nonpeptide HIV-1 protease inhibitor: elucidation of the binding modes and its application in the design of related analogs".

Vara Prasad, J. V. N., Para, K.S., Ortwine, D. F., Dunbar, Jr., J. B., Ferguson, D., Tummino, P. J., Hupe, D., Tait, B. D., Domagala, J. M., Humblet, C., Bhat, T. N., Liu, B., Guerin, D. M. A., Baldwin, E. T., Erickson, J. W. and Sawyer, T. K. (1994) *J. Am. Chem. Soc.*, **116**: 6989-6990. "Novel series of achiral, low molecular weight, and potent HIV-1 protease inhibitors".

Stoddard, B. L. and Koshland, Jr., D. E. (1993) *Proc. Natl. Acad. Sci. USA* **90**: 1146-1153. "Molecular recognition analyzed by docking simulations: The aspartate receptor and isocitrate dehydrogenase from Escherichia coli".

Jeffery, C. J. and Koshland, Jr., D. E. (1993) *Protein Science* **2**: 559-566. "Three-dimensional structural model of the serine receptor ligand binding domain".

Goodsell, D. S., Lauble, H., Stout, C. D. and Olson, A. J. (1993) *Proteins: Str. Func. and Genet.*, **17**: 1-10. "Automated Docking in Crystallography: Analysis of the Substrates of Aconitase".

Stoddard, B.L. and Koshland, Jr., D.E. (1992) *Nature*, **358**: 774-776. "Prediction of a receptor protein complex using a binary docking method".

## V 4. Web publications

Lunney, E. (1995) *Network Science*, **1**: http://www.netsci.org/Science/Cheminform/feature01.html. "Structure-Based Design and Two Aspartic Proteases".

Appendices