Computational studies of *E. coli* DHFR: Drug design, dynamics, and method development

by

Michael G. Lerner

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Biophysics)
in The University of Michigan
2008

Doctoral Committee:

Associate Professor Heather A. Carlson, Chair
Professor Daniel M. Burns, Jr.
Professor Gordon M. Crippen
Associate Professor Nils G. Walter
Assistant Professor Ioan Andricioaei

Acknowledgements

I would like to thank my advisor Dr. Heather Carlson for her tireless support throughout my graduate career. I entered the University of Michigan knowing only that I wanted to think like a physicist, but about biological problems. She has guided me through the long and twisting path from that desire to the completion of my degree and the start of my professional career. Not only has she provided me with the opportunity to spend my working hours focusing on things I love, but she has been an outstanding mentor, as well as being a great source of both professional and personal support, even in the hardest of times.

I would like to thank all of the members of the Carlson lab throughout my time at Michigan. The lab itself is a great community, and one member is always willing to help out another lab member. In particular, I would like to thank Dr. Kristin Meagher, who took a large role in bringing me up to speed in the group and provided endless hours of interesting scientific discussion. I would also like to thank Dr. Kelly Damm, with whom I discussed all of my projects in great detail. Mark Benson and Richard Smith deserve a special mention for the time we spent together on the BindingMOAD project. Allen Bailey was an excellent system administrator, keeping our computers up and running despite a myriad of hardware and software issues.

I would like to give a special thanks to the members of my dissertation committee for helping me learn exactly what goes into a thesis. I would especially like to thank Dr. Nils Walter, with whom I did my first rotation and who has been watching my career from the beginning of graduate school. I would like to thank Dr. Gordon Crippen for his detailed theoretical knowledge of computational chemistry and help with MOE and SVL. I thank Dr. Ioan Andricioaei for his consistent insights into every research topic I have asked him about. I thank Dr. Dan Burns for his level headedness and help steering me in the right direction.

Several thanks are also due to people who are not a part of the University of Michigan, including Dr. Warren DeLano for his help and discussions relating to PyMOL. Without PyMOL, my time as a graduate student would have been significantly less productive and enjoyable. Dr. Walter Smith first allowed me to see the excitement of biophysics as an undergraduate student, and his advice has carried me through to this day. Ruby Hogue and Lynn Alexander were both fantastic secretaries and sources of professional support.

Finally, I would like to thank my family and friends. Thank you to my loving wife, Dr. Heather Lerner, without whom I never would have succeeded. You are an inspiration to me, both personally and professionally. Thank you to my parents David and Esther Lerner, who have taught and supported me from the first day that I can remember. My wife's parents Neill and Fannie Kiley have also been a great source of support and encouragement. Thank you to all of my friends. My graduate career has had more than its fair share of ups and downs, and you made the ups fun and the downs bearable.

I was fortunate enough to be supported for several years by the Molecular Biophysics Training Grant (NIGMS grant GM008270), and the extra opportunities provided by that grant are just as gratefully acknowledged as the funding.

Table of Contents

List of Figures

List of Tables

List of Appendices

Abstract

Computational studies of *E. coli* DHFR: Drug design, dynamics, and method development

by

Michael G. Lerner

Chair: Heather A. Carlson

Dihydrofolate reductase (DHFR) catalyzes the NADPH-dependent reduction of dihydrofolate to tetrahydrofolate. As the only source of tetrahydrofolate (an important precursor in the biosynthesis of purines, thymidylate, and several amino acids), it has been a long-standing anti-cancer target and a classic system for structure-based drug design (SBDD). Escherichia coli DHFR (ecDHFR) is a canonical system for studying enzyme structure, dynamics, and catalysis. Protein flexibility and dynamics are of utmost importance in understanding the structure and mechanism of DHFR. This has been well investigated computationally and experimentally. The conformation of the M20 loop is particularly important to the catalytic cycle, as its three major conformations (open, closed, and occluded) are known to regulate ligand affinity and turnover. In addition to these static conformational differences, correlated dynamics are known to be of primary importance, showing distinct changes during different stages of the catalytic cycle. The dynamics have been used to explain the effects of distal mutations.

We have performed two 10-ns molecular dynamics simulations of the ecDHFR•NADPH complex. We discovered transient, sub-nanosecond, correlated dynamics that correspond to correlations found in the catalytically active state. These dynamics involve both the protein and the cofactor. We found conformational changes that clearly indicate preorganization of the binding site related to folate binding. We have also discovered a potential new allosteric site, supported by extensive computational work as well as by crystallographic and mutagenesis results in the literature.

Traditional SBDD techniques focus on static structures. In 1999, Carlson and coworkers introduced the MPS (multiple protein structure) method as a way of incorporating protein flexibility into SBDD. The extreme importance of flexibility for DHFR makes the MPS method particularly appropriate. To improve the method, we developed new techniques for flooding and automatically clustering the solvent-mapping probes used in the procedure. We generated models from simulations starting with the M20 loop in both open and closed conformations. The MPS models preferentially identified high-affinity inhibitors over drug-like non-inhibitors.

Chapter 1

Introduction

Static protein-ligand binding interactions

An understanding of protein recognition is the basis for our investigations into drug design, dynamics, and methods development. Emil Fischer set the stage for investigations in this area with his seminal "lock-and-key" hypothesis in 1894.[1] He proposed that the 3D surfaces of binding partners must be complementary with the substrate fitting into the enzyme like a key into a lock. Although this analogy does not provide the full story of protein binding events, it has greatly shaped our understanding. A large amount of current data indicates several common features across protein-ligand systems, summarized recently by Böhm[2] in three main points: (1) Emil Fischer's lock-and-key binding theory is generally important, as the shapes of ligands and their binding cavities are often complementary; (2) direct, complementary interactions between ligands and their binding cavities (hydrogen bonds, ionic interactions, hydrobophic interactions, etc.) are important and repulsive interactions (e.g. close contact between two positively-charged side chains) are rare; and, (3) while the conformation of a bound ligand is not required to be an energetic minimum, it is typically energetically favorable.

Most protein-ligand binding events involve non-bonded interactions.[2,3] The forces involved in direct protein-ligand (and protein-protein) interactions span several orders of magnitude and extend from close-range (r^{-12} for repulsive van der Waals) to long-range

(r^{-1} for charge-charge) interactions. The longer-range forces can be particularly important in bringing proteins and ligands close to each other, a prerequisite for binding, whereas all of these forces are important for the actual binding. The binding affinity of a protein-ligand complex is determined by the change in free energy associated with their interaction.[4] The main forces involved are direct forces such as Coulombic interactions, hydrogen bonding, van der Waals, as well as entropic and hydrophobic forces.[1-3,5-8] A recent review summarizes the magnitudes of these forces,[9] and a typical computational force field used to model these interactions may be seen below in the Molecular mechanics section.

Computational investigations of binding free energies are complicated by the fact that they rely on determining the difference between two large numbers, the interaction energy of the protein and the ligand and the interaction energy of both with water.[7,10,11] Since those two numbers are both much greater than their difference, it is particularly important to reduce errors in the calculation of either.[12] Protein-ligand interactions are frequently modeled with force fields as mentioned above, and physics-based free-energy calculations are often used to calculate protein-ligand binding affinities.[13]

An extremely interesting, but less popular method, involves grand canonical Monte Carlo simulation, a technique demonstrated on thermolysin and T4 lysozyme.[14] In this method, a series of grand canonical ensembles is generated, each containing many ligands interacting with a protein. For each system, a quantity related to both the chemical potential and the concentration of particles in the system is annealed. Unlike other standard statistical mechanical ensembles, the grand canonical ensemble allows the number of particles to vary. During the annealing, Monte Carlo sampling (see section below) is used to both move ligands and to cause them to appear or disappear. The annealing process is preformed at decreasing free energies, and the sampling allows binding free energies to be calculated across the entire protein surface.

With respect to interactions with water, the details of the hydrophobic effect have been heavily studied, and three reviews provide an excellent summary of the current understanding of the topic.[15-17] The hydrophobic effect is of key importance in protein folding, structural stability, binding, and protein-membrane interactions. The highly cooperative entropic changes involved are still not fully understood, and a range of theoretical models, some extremely complex and some quite simple, is being developed to provide insight. There are many strategies presently used to model hydration in molecular simulations, ranging from scoring functions to detailed free energy calculations. A review by Mancera summarizes several of these methods, pointing out recent advances that have led to efficient algorithms and faster calculations.[10]

Multiple conformations

Having considered the forces involved in protein-ligand interactions, it is worth considering the conformation of the protein. Emil Fischer's lock-and-key binding theory does hold in some cases. For example, a conserved hydrophobic lock-and-key interaction is responsible for anchoring two glutathione transferase A1-1 subunits,[18] and rigid lock-and-key models were shown to be surprisingly effective in classic protein-ligand docking studies.[19] In 1930, Haldane improved on the lock-and-key theory by suggesting that enzyme-binding sites might be used to bring substrates together or to exert forces directly upon them.[1,20,21] Pauling refined this concept in 1946 stating that, in contrast to the case of antibodies and antigens where one can expect the two molecules to be completely complementary in their native states, an enzyme should actually be complementary to a ligand's *transition state*.[22] However, more recent work has shown that, even in the case of antibody-antigen interactions, this model may not quite be sufficient.[23,24]

The preceding theories generally consider the protein to be in a single, static conformation. There have been two major modern improvements to this theory that account for protein movement: the induced fit hypothesis and the conformational selection hypothesis.[25] In 1958, Koshland proposed the "induced fit" model in which an enzyme undergoes conformational changes as a result of ligand binding.[26,27] Although several different types of molecules may be able to bind in an active site, only the appropriate ligand can induce the particular conformational changes required for catalysis. The induced fit model was originally proposed as an explanation for enzyme-substrate systems, but the central idea of a flexible protein has been applied to our understanding of many types of interactions including protein specificity, regulation and cooperativity.[27] The conformational selection hypothesis (also called the pre-existing equilibrium hypothesis) suggests that the native state of a protein is actually made up of an ensemble of accessible conformations, all of which co-exist with some probability distribution.[23,28-30] The equilibrium ensemble is generally considered to be comprised of states that, while exhibiting conformational diversity, are close in free energy. A ligand interacts with this ensemble by binding to the conformation that it best complements, thus shifting the equilibrium of states. In this manner, a productive ligand may bias the entire ensemble towards active conformations, while an inhibitor can lock a protein into an unproductive state. In this view, ligands actually "see" several binding sites of different shapes and sizes, helping to explain how a single protein can show high affinity for binding multiple ligands.[29] In the following section on allostery, we will see how binding outside of the active site can affect the ensemble. It is also worth noting that proteins in different stages of a catalytic cycle may be described by different conformational ensembles.

Allostery

On the other hand, sites other than the active site may be involved in protein regulation. For instance, allosteric regulation involves a molecule (called an effector or a modulator) binding reversibly and non-covalently to a site on the protein, other than the active site.[1] This interaction can regulate the activity of the protein. At its heart, allosteric regulation is simply the transmission of information across a protein, initiated by the binding of an effector molecule. In homotropic regulation, the effector is the same type of molecule as the substrate. In heterotropic regulation, they are different. The MWC model, one of the first successful explanations of allosteric regulation, was proposed by Monod, Wyman, and Changeux in 1965.[31] The model was proposed for hemoglobin, which is comprised of four identical subunits. Each subunit could be in two conformational states, R (relaxed) and T (tense), both of which have a different ligand affinity. The MWC model is strictly symmetric, and all subunits are in the same conformation, either all-R or all-T. The binding of successive effectors increases the probability of an overall transition to the high-affinity state.

The KNF model, proposed shortly thereafter by Koshland, Nemethy, and Filmer in 1966, allows for individual subunits to undergo conformational change separately.[32] Substrate binding increases the likelihood of conformational change in that subunit, and thus, in neighboring subunits. Therefore, the MWC model can be viewed as the limiting case of the KNF model. Although the topic has been examined extensively in subsequent years, the precise details of the mechanisms of allosteric regulation are still not fully understood.[1,9,33-35] However, it is understood that ligand binding at one site can induce conformational changes at another site and along a pathway connecting the two sites.[36] These changes can occur over very large distances. In the case of the aspartate receptor, a ligand-induced change of approximately 1 Å is propagated via a piston-like response over a distance of around 100 Å, leading to a dramatic signaling-cascade response.[37]

While there were only 24 allosteric proteins to investigate in the original MWC study,[31] more recent studies indicate that there are at least hundreds of allosteric proteins, encompassing several different protein classes and providing numerous new drug targets.[33,38] Our understanding has even grown to the point where allostery can be engineered. For instance, while Pyruvate kinase M_1 is a non-allosteric enzyme, a single-residue mutatation, Ala398Arg, transforms Pyruvate kinase M_1 into an allosteric protein.[39] Another group, starting from a non-allosteric phosphofructokinase, was able to engineer an allosteric enzyme via a series of C-terminal deletions and mutations.[40]

Throughout the intervening decades since the MWC model, our definition of allostery has changed. Historically, the term "allosteric" was reserved for symmetric, oligomeric proteins.[41] More recently, it has been applied in general to systems where one ligand's binding affinity is affected by the binding of another molecule, even in the case of monomeric proteins.[33,35,42]

Moreover, the fundamental assumption that allosteric proteins shift discretely between two distinct states has been challenged. In 1972, Webber demonstrated that a simple two-state model was not sufficient to explain observed behavior. He proposed that allosteric proteins exist in a dynamic equilibrium of states and that the binding of an effector induces a shift in the conformational ensemble occupied by the protein.[43] The current mainstream understanding of allostery is, indeed, that of a ligand-induced change in the makeup of a protein's conformational ensemble [35]. Nussinov and co-workers have taken this argument one step further. Since all proteins exist in such a conformational ensemble, and allostery is simply a shift in this ensemble, they suggest that all proteins may contain the potential for allostery.[33] More specifically, they make the distinction between dynamic, stiff, and fibrous proteins, arguing that all dynamic proteins have the potential to be allosteric. They argue that dynamic proteins thus provide potential novel drug targets, as a shift in the protein's overall conformational ensemble has the potential to alter the binding site's conformational ensemble. In this view, allosteric effects are not so fundamentally different from those seen in systems where distal motions and mutations modulate active-site motions and activity. This is seen in systems such as β-1,4-galactosyltransferase, where mutations in one small loop are known to affect the flexibility of the larger functional loop,[44] or dihydrofolate reductase (DHFR), where several distal mutations have been shown to have strong effects on catalysis.[45-61]

Some studies propose an even more drastic departure from the MWC model: the dynamic motions of the proteins may, in and of themselves, be responsible for allosteric regulation. As early as 1984, Cooper and Dryden proposed a model by which effector binding could induce allosteric effects purely by altering the frequencies and amplitudes of protein fluctuations.[62] This process is primarily related to entropy and does not require conformational change at all. In 2006, the cAMP/CAP system was used as the first demonstration of the existence of this purely dynamics-driven allostery.[63] They note that experiments have shown that proteins contain a significant amount of residual entropy[64-66] and that this entropy is quite important for biological activity, indicating that dynamics-driven allostery could actually be quite prevalent in nature.

A careful study in which 32 10-ns molecular dynamics (MD) simulations were used to study allostery in the chemotaxis Y (CheY) protein provides some resolution on the topic: they show that no single definition of allostery (MWC, conformational shift, dynamical shift, etc.) is sufficient, and results must be explained via a combination of all of these allosteric mechanisms.[34] They conclude that one should study collective protein motions, and they do so via normal mode analysis.

In Chapter 3, we study the collective motions of DHFR via correlated dynamics, and we discover a new potential allosteric site. In Chapter 4, we present PyPAT, a suite of tools that assist in the computational examination of these characteristics.

Protein motions and experimental techniques

Of course, this ensemble of conformations is brought about by an ensemble of protein motions. Protein motions have been studied on scales ranging from femtoseconds (bond vibrations), through pico- and nanoseconds (side-chain reorientation, loop movements), up to seconds (breathing motions, subdomain interactions).[64,67,68] NMR experiments provide a powerful approach for studying protein flexibility, covering a wide range of time scales.[69] Hydrogen/deuterium exchange experiments have been used to demonstrate directly that native protein states consist of ensembles, rather than single static conformations.[70] These effects may be seen to a surprisingly large degree, even for thermophilic enzymes.[71] Small-angle scattering also provides a means to study an entire conformational ensemble and to study changes in that ensemble.[72] X-ray crystallography can also be used to study protein conformations in different static states and conditions[73] and can provide particularly important insight under time-resolved conditions with nanosecond-level[74] or picosecond-level[75] resolution. Experiments at all of these levels, as well as computational investigations, are necessary to build up a complete picture of protein conformation, function, and dynamics.

Several studies show that the motions themselves are of direct importance. A recent normal-mode analysis of the globin family of proteins has found that the entire family shares common dynamics and that these collective motions are of critical importance to the biological function of the globins.[76] NMR experiments on CypA have demonstrated the importance of dynamics for catalysis, indicating a strong connection between the rates of conformational dynamics and those of turnover.[77] NMR studies on DHFR have shown that dynamical flexibility is key to catalysis.[57,78] Subsequent work on CypA has shown that pre-catalytic conformational ensembles include states that are directly relevant to catalysis.[79] These studies highlight our overarching theme: proteins must be understood in terms of conformational and dynamical ensembles, rather than as static systems. This modern, integrated view of flexibility and dynamics can help to explain not just binding and allostery, but also protein folding and enzyme pathways.[67,68,80-82]

Computational methods of investigating protein motions

Molecular mechanics (MM)

Many computational techniques are available for the study of biological systems. The most accurate of these is quantum mechanics. A full quantum mechanical model captures the true physics of the system, but such models are too computationally intensive for medium to large biological systems.[11] Thus, various levels of approximations must be made, and so molecular mechanics (MM) force field techniques were developed. MM techniques use Newtonian mechanics to approximate the forces involved in a system, and a typical model includes terms for bond stretching, bond-angle bending, dihedrals, and non-bonded terms (electrostatic and Van der Waals). The Born-Oppenheimer approximation is made, and electronic structure is ignored. Polarizability is also usually ignored, and electrostatics are modeled via point charges. Simple harmonic oscillator models of bond stretching and angle bending are used, and dihedrals are treated via a cosine expansion. The current AMBER force field,[83] a fairly typical example, is shown below:

$$U(R) = \sum_{bonds} K_r (r - r_{eq})^2 + \sum_{angles} K_\theta (\theta - \theta_{eq})^2 + \sum_{dihedrals} \frac{V_n}{2}(1 + \cos[n\phi - \gamma])$$

$$+ \sum_{i<j}^{atoms} \left(\left(\frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} \right) + \frac{q_i q_j}{\varepsilon R_{ij}} \right)$$

K_r are bond-stretching constants, r are bond distances, r_{eq} are the equilibrium bond distances, K_θ are the angle-bending constants, θ are bond angles, and θ_{eq} are the equilibrium bond angles. V_n are dihedral force constants, ϕ are the dihedral angles, and γ are the dihedral phase angles. Non-bonded interactions are modeled with a 6-12 Lennard-Jones potential and simple point-charge interactions. The Lennard-Jones potential is an approximate model for several short-range forces including attraction due to induced dipole-dipole interactions and repulsion due to electrostatics and the Pauli exclusion principle. In the 6-12 potential, the longer-range attractive force constants are B_{ij}, the shorter-range repulsive force constants are A_{ij}, and the interatomic distances are R_{ij}. For interactions between point charges, and a simple Coulombic potential is assumed where ε is the dielectric constant (1 in the case of explicit solvent) and q_i is the charge on atom i.

Given the high degree of approximation used in such a force field, great care must be taken in determining appropriate parameters for a system. Individual terms are compared against many experimental properties as well as high-level quantum-mechanical calculations.[11,84] Many MM models enjoy widespread use, some of the most popular including AMBER,[83] CHARMM,[85] and OPLS[86].

Molecular dynamics (MD)

MD techniques use the force field provided by an MM model and Newton's second law (F = ma) to provide a trajectory for the systems of interest.[11] Current MD techniques are typically used to examine systems on the order of tens to hundreds of nanoseconds,[11,87] although some MD trajectories have been calculated at the microsecond scale[88,89] and the newest algorithms and hardware[87,90] are capable of millisecond-scale runs. It is worth noting, however, that the accuracy of an MD simulation is directly limited by the accuracy of the underlying MM force field. MD simulations performed with AMBER[83] were used to generate the data analyzed in Chapters 3 and 4.

There is not a single, standard, generic approach to analyzing MD trajectories. The details of the individual simulation and the questions of interest to the simulator must be taken into account in each case. First, one must verify that the system has been properly equilibrated. MD simulations require an initial set of coordinates and velocities. Structures are almost always taken from X-ray or NMR data and velocities are taken from a Maxwellian distribution. While these may represent a favorable structure under the experimental conditions, MD simulations use a different potential than those conditions. This means that the structure must be equilibrated under the MD potential. In particular, crystal-packing forces can push side chains into unfavorable conformations. In the case of DHFR, crystal-packing forces completely control the conformation of a key loop.[91] MD codes output information including temperature, total energy, potential energy, and other individual energetic contributions from the force field at user-defined intervals throughout a simulation. These parameters should be examined to ensure convergence and stability. In order to verify that the structure itself is stable, one typically calculates the root mean square deviation (RMSD) between trajectory snapshots and a reference set of coordinates (typically a crystal structure, a snapshot from within the MD trajectory, or an average structure from the trajectory). Convergence of these RMSDs indicates structural stability. Although many systems are fully equilibrated after several hundred picoseconds of simulation (e.g. HIV-1 protease[92]), systems such as DHFR may take more than 1 ns.[45,93] The data generated after the system has been equilibrated is called production data, and the term "trajectory" typically refers only to production data.

After verifying that the system has been equilibrated, there are several ways in which one may continue to examine the trajectory. The type of analysis performed is guided by the specific questions in which the researcher is interested. Fluctuations of individual residues may be studied via RMSDs, bond and dihedral angles, etc. Secondary structure can be evaluated throughout the simulation to show, for example, the lengthening and shortening of loops and helices, the creation and destruction of hydrogen bonds, or the fraction of total native contacts. A classic study by Duan and Kollman[88] used a native-contact and secondary-structure analysis to examine the fundamental nature of protein folding. As the full sets of coordinates are available throughout the entire simulation, the user has great flexibility in this area. Specific quantities are often defined to address particular questions (e.g. various combinations of distance and angle measurements may be used to obtain a more precise measurement of hinge-bending[94]). The intrinsic motions of the system may also be studied via an array of techniques such as normal mode analysis,[95,96] essential dynamics,[97-99] and correlated dynamics (see section below entitled “Correlated dynamics”). With such a wide variety of analysis techniques available, and with the ability to code new methods when necessary, a creative approach can allow the researcher to examine an extremely wide range of questions via MD.

Correlated dynamics

Covariance and cross-correlation techniques are frequently used to examine the relative motions of different atoms throughout a protein, and can be applied to the study of normal modes as well as MD trajectories.

Given a vector X whose components are random variables (x_1, … , x_n) with means (μ_1, … , μ_j) and standard deviations (σ_I, … , σ_j), the covariance of two entries, x_i and x_j is defined as the expectation value

$$\mathrm{Cov}(x_i,x_j) = < (x_i - \mu_i)(x_j - \mu_j) >.$$

The resulting covariance matrix, whose (i,j)th entry is $\mathrm{cov}(x_i,x_j)$ is thus a symmetric, positive definite matrix. The magnitude of the covariance depends on the standard deviations, and is therefore often scaled to obtain the correlation:

$\mathrm{Corr}(x_i,x_j) = \mathrm{cov}(x_i,x_j) / (\sigma_i\sigma_j)$.

The correlation matrix is formed in the same way as the covariance matrix, and is symmetric and positive definite. The Cauchy-Schwarz inequality[100,101] can be used to show[102] that the correlation ranges from -1 to 1. Two elements that vary in sync with one another will have a correlation of 1, while two perfectly anti-correlated elements will have a correlation of -1, and two independent variables will have a correlation of 0.

These methods were originally applied to BPTI, where they were shown to provide insights that were relatively independent of simulation methodology (MD, etc.) and environment (implicit and explicit solvent).[103] More recently, correlated dynamics have been used to investigate DHFR, where it was found that strong coupled motions are present in the reactive complex (DHFR•DHF•NADPH), but are absent in complexes from other stages of the catalytic cycle.[45] As might be expected,[103] strong positive correlations between pieces of secondary structure are found throughout all of their simulations. Though these positive correlations do change throughout the complexes, it is the differences in anti-correlated motions that are particularly striking. Several catalytically important mutations are investigated, and those not directly involved in catalysis are found to occur in these anti-correlated regions. A follow-up study on four of these mutations shows that they disrupt the strong coupled motions.[104] These data strongly suggest that correlated dynamics may be coupled to catalysis itself. Correlated dynamics have also been used to examine binding effects in systems such as cAMP/CAP, where it has been shown that cAMP binding to one CAP monomer can affect the dynamics of the other (unbound) monomer.[105] In Chapter 3, correlated dynamics are applied to the study of collective motions in DHFR•NADPH.

Monte Carlo (MC)

In MD, an initial velocity distribution is assigned and a force field is used to calculate subsequent conformations. Instead of providing a time-varying trajectory of states, MC (credited to Ulam and von Neumann[106,107] and named in honor of a famous casino in Monaco) works by using random sampling to produce a collection of configurations. Metropolis MC[108] generates these states as a series of "moves." Given an initial system configuration, a random process is used to generate a potential new configuration. If the potential energy of this state is lower than that of the preceding configuration, the move is accepted. If it is higher, a random number ξ is chosen between 0 and 1. If ξ is less than the Boltzmann factor ($e^{(-\Delta E/kT)}$, where E is the potential energy of the system, k is Boltzmann's constant, and T is the temperature), the move is accepted. If it is greater, the move is rejected and the old configuration is used as the next state. This series of moves, which technically forms a Markov chain,[11] can be an efficient way of exploring conformational space. While MD requires the calculation of forces, MC only requires potential energy to be calculated.

It is worthwhile to note the effect of temperature on this process. The system temperature directly controls the probability of accepting a higher-energy move. Thus, a high-temperature simulation will sample more high-energy states, but a low-temperature Metropolis MC run will tend to steer the system towards a local minimum. This technique, implemented with OPLS[86] force field and a modified[109] version of BOSS,[110] is used to minimize small-molecule probes in our drug design work (see Chapter 2, Chapter

5, and the sections below entitled drug design and our results). MC methods have been broadly applied to the study of proteins including the investigation of protein folding via global MC minimization,[111] the calculation of generic thermodynamic properties, and simulated annealing.[11]

Calculating allostery

General techniques used to study protein motions and conformations can be applied to the study of allostery, as highlighted by the correlated dynamics studies on DHFR mentioned above. In addition to these, two fascinating techniques have recently been developed specifically for the study of allostery. Ming and Wall[112] have developed a theoretical technique for the quantification of allosteric effects in proteins, taking the new ensemble-based definition of allostery into account. $P(\mathbf{x})$ is defined as the equilibrium probability distribution of apo-protein conformations $\mathbf{x}$, and $P'(\mathbf{x})$ is defined as the ligand-bound distribution. They then use the Kullback-Leibler[113] divergence, $D_{\mathbf{x}}$, as a measure of the difference between the probability distributions and argue that it is a reasonable metric for quantifying allosteric effects. The harmonic approximation is used in their derivation of $D_{\mathbf{x}}$, and normal mode analysis is then used to examine lysozyme binding, verifying their claims that large values of $D_{\mathbf{x}}$ correspond to known allosteric interactions.

Anisotropic thermal diffusion (ATD) was developed by Ota and Agard to study intramolecular signaling in general.[114] ATD is a computational molecular dynamics technique in which the protein is initially minimized and subjected to an extremely low-temperature (10 K) equilibration. A specific subset of residues is then heated to 300 K. In the absence of any sort of signaling pathways, the heat from these residues is expected to propagate isotropically. Evidence of mechanical pathways for allosteric communication is found when the heat propagation is anisotropic. This technique has yielded consistent results when probing both ends of an allosteric path in the PDZ protein PSD-95.[114] In Chapter 3, ATD is used to examine a possible allosteric pathway in DHFR.

Structure-based drug design (SBDD)

Structure-based drug design refers to the process of using three-dimensional information about protein and ligand structure to aid in the drug-design process. These structures typically come from X-ray crystallography or NMR experiments. Although the most widely used SBDD techniques have used static structures, it is becoming increasingly well understood[115] that proper incorporation of conformational flexibility is of prime importance to structure-based drug design and discovery. SBDD techniques can be broadly classified as ligand-based or receptor-based.[116] Ligand-based methods generally start with a known set of inhibitors and search for other similar molecules, often using pharmacophores or 3D shape matching to screen databases of drug-like molecules.[116,117] Receptor-based methods focus instead on the shape and chemical features of the receptor in the attempt to find new binding partners. The SBDD efforts of this dissertation use receptor-based methods.

Although many computational techniques are available to incorporate ligand flexibility, programs that incorporate *protein* flexibility are few and far between. This is a very active field of research.[115,118,119] The main focus of the SBDD work in this dissertation is the multiple protein structures (MPS)[109] method. In order to understand protein function, specificity, and inhibition, researchers will often create a reciprocal map of a protein surface. Rather than asking questions directly about the protein, such a map answers the question “what features would a binding partner contain?” Multiple-copy methods (MCMs) use small-molecule probes to generate these maps. A protein’s surface is flooded with hundreds of these probes. The probes are then independently minimized to the surface. Different types of probes map out different types of chemical functionality (hydrophobic, hydrogen bonding, ionic, etc.), and clusters of minimized probes reveal the most important and biologically accessible interaction sites. The MPS method is a particular type of MCM in which protein flexibility is incorporated by repeating this process on an ensemble of structures, looking for consensus across the ensemble (i.e., clusters of probes in the same location in many of the protein’s conformational states). Ensemble structures may be taken from NMR, X-ray, or MD studies.[120] The technique has enjoyed considerable recent success on HIV-1 protease,[92,120,121] DHFR,[93,122] and p53/HDM2.[123] It has been shown to predict new chemical scaffolds for drugs, as well as to predict binding partners from *apo* structures. We apply the MPS method to the study of DHFR in Chapter 2.

Results from this work

The original goal in this dissertation was to apply the MPS method to the study of dihydrofolate reductase. DHFR catalyzes the NADPH-dependent reduction of dihydrofolate (DHF) to tetrahydrofolate (THF).[124] DHFR is the only biological source of tetrahydrofolate, an important precursor in the biosynthesis of purines, thymidylate, and several amino acids.[125,126] Thus, it has been a long-standing anti-cancer target[127] and a classic system for SBDD.[126,128] DHFR inhibitors have been used to treat human diseases including psoriasis, autoimmune diseases, and neoplastic diseases, and the potential to target species-specific DHFR has also made it an important antibiotic and anti-microbial target.[126,128]

The structure, dynamics, and function of DHFR have been studied extensively, as summarized in a recent review.[128] Multiple conformational states have been identified through X-ray crystallography.[91] Flexibility and sampling between these conformations are of paramount importance throughout the catalytic cycle. The role of protein dynamics in DHFR catalysis has been well studied both with calculations[45,81,129] and experiments, particularly NMR spectroscopy.[130-132] DHFR is composed of an eight-stranded β-sheet (βA - βH) and four α-helices (αB, αC, αE, αF). It has been divided into two rigid-body subdomains. The first (the adenosine-binding domain) consists of βB, βC, βD, βE, αC, αE, αF, and the intervening residues. The second (the loop subdomain) consists of the rest of the structure, and is formed mostly by loops,[91] including the CD, FG, GH and M20 loops. All regions of DHFR contribute dynamically to substrate binding and chemical turnover.[133] Correlated dynamics, as discussed above, have been shown to contribute to catalysis and to change in distinct patterns throughout the catalytic cycle.[45] The conformation of the M20 loop is particularly important to the catalytic cycle, as it is known to regulate ligand affinity and turnover.[58,78,91,128,134-138] Three conformations are reported in the crystallographic literature: open, closed, and occluded. The Michaelis complex DHFR•NADPH•DHF is found with the M20 loop closed. The closed form is also the only conformation seen in DHFR from all species, regardless of the crystal packing and ligands bound in the substrate and cofactor binding sites.[91] When the product is bound (DHFR•NADP+•THF, DHFR•THF, DHFR•NADPH•THF), the M20 loop is found in the occluded position. When a cofactor is present, the occluded loop forces its nicotinamide moiety out of the pocket. The closed loop is a conformational intermediate between the extremes of the open and occluded loops. The DHFR•NADPH complex is found in both the open and closed conformations.

Escherichia coli DHFR (ecDHFR) is a canonical system for studying enzyme structure, dynamics, and catalysis. Fierke *et al.* determined the full catalytic cycle,[139] summarized in the first figure of Chapter 3. In biological systems, DHFR is always found with at least one binding partner, a cofactor or a ligand in the folate-binding site, and DHFR•NADPH is the analog of an *apo* state. This dissertation follows our studies of the DHFR•NADPH system as well as describing several of the tools developed in those studies.

Our studies are primarily based on two MD simulations of the DHFR•NADPH complex, one starting from a closed-loop crystal structure and one starting from an open-loop crystal structure. Once we had obtained an ensemble of structures from those simulations, the MPS method requires us to flood the active site with a series of small-molecule probes. Previous applications of the MPS method had focused on systems such as HIV-1 protease that have fairly open active sites. DHFR's active site is much more closed off, and it is known that important ligand-binding interactions are made at the bottom of the binding site.[91] In Chaper 5, we develop new flooding techniques that allow us to properly sample both open and closed active sites.

After the probes had been placed and minimized, we were faced with the task of grouping them into clusters. A vast array of techniques are available for clustering data, and several excellent books[140,141] have been written on the subject. Previous MPS studies have used "by-hand" visual inspection to group probes into clusters. This method has the advantage that it has been thoroughly tested, and it works. It has several disadvantages: it is very time-consuming, taking approximately two to three weeks for a typical system. It requires training and can be subjective. More troubling is the fact that visual recognition is limited to a maximum of approximately 10-15 snapshots, after which there is simply too much data to separate clusters cleanly. In Chapter 5, we develop an automated probe-clustering method based on Jarvis-Patrick[142] clustering. The validity of this method is investigated via the study of protein-protein interfaces, and it is shown to be fast, reproducible and robust. Results are then favorably compared to previous MPS results[121] on the HIV-1 protease system. The method is generally applicable to the clustering of small molecules in three-dimensional space, and all of the details are spelled out explicitly in the hopes that other groups will see its utility.

At this point, we have the tools to apply the MPS method to the study of DHFR, and so in Chapter 2, we develop receptor-based pharmacophore models of the DHFR•NADPH complex to examine the implications of flexibility for SBDD on this classic system. We show that receptor-based pharmacophore models for SBDD generated from the closed-loop structure perform better than those generated from the open-loop structure. However, we find that the incorporation of increased flexibility into the open-loop models greatly improves their performance. Notably, the inclusion of greater flexibility (in both models) does not result in a loss of specificity, as the optimal models preferentially identify potent ecDHFR inhibitors over other general DHFR inhibitors.

After obtaining these results, we begin an examination of DHFR's flexibility and dynamics in Chapter 3. We investigate both conformational changes and correlated dynamic motions. Recent NMR studies have shown directly that the conformational ensemble experienced by DHFR in any given state along the catalytic pathway includes structures from nearby catalytic states.[143] Unfortunately, the NMR experiments are unable to provide the full structural details comprising these states. Our MD studies of DHFR•NADPH are able to provide such details, and we find several clear examples of binding-site pre-organization (including loop conformations and binding-site residue orientations) even though no ligand is bound in the active site.

Correlated dynamics have been shown to contribute to catalysis and to change in distinct patterns throughout the catalytic cycle.[45] MD simulations of the Michaelis complex have revealed strongly correlated and anticorrelated motions involving distant regions of the protein.[45] This extensive coupling was only seen in the reactant form, not in either DHFR•NADP$^+$•THF (product ternary complex) or DHFR•NADPH•THF (product release complex), implying that these motions may be necessary for catalysis.[45] Thus, we examined the correlated dynamics of the DHFR•NADPH complex. We find that the strong correlated dynamics from DHFR•NADPH•DHF appear transiently throughout our simulations, existing prior to the binding of DHF. Just as proteins sample a conformational ensemble of states, only some of which are relevant to ligand binding and catalysis, our results indicate that they sample a similar ensemble of correlated dynamics, the makeup of which changes throughout the catalytic cycle.

Finally, we turn our eye toward the bottom of the folate-binding site. We find that it opens up throughout our simulations, and it leads us to the discovery of a cavity and new potential binding site on the far side of DHFR. Several computational techniques are used to show that this new site has a high potential to bind ligands. Mutagenesis results from the literature indicate the possible allosteric properties of this site, and ATD is used to determine a putative pathway for allosteric communication with the folate-binding site.

The development of computational tools necessary to implement these studies is presented in Chapter 4.

Potential impact

The results of this dissertation have had large impacts within the Carlson lab and will hopefully contribute greatly to the study of protein interactions. One of the major focuses of the Carlson lab is the development and application of the MPS method. The results from Chapter 2 serve as further validation of the MPS methodology itself. The scripts developed throughout Chapters 2-5 automate, refine, and speed up the MPS method, allowing it to be quickly, objectively, and reliably applied to new classes of proteins. The time required to cluster probes from individual snapshots has been reduced from the scale of weeks to that of hours without sacrificing the quality of results. DHFR was significantly different in many ways from previous systems studied via MPS, (shape and dynamics of the active site, overall system charge, etc.) and the techniques developed herein greatly expand the overall applicability of the MPS method. Efforts to more fully automate the MPS method are currently underway.

Several of the techniques developed here have broad application outside of our MPS work. There are a wide variety of computational packages such as MCSS,[144] MCSS2PTS,[153] and LUDI[145] that require the clustering of small probe molecules. Some authors have noted the need for better clustering algorithms in such cases,[146,147] and it is our hope that the methods developed in Chapter 5 will be incorporated throughout the field, as they are applicable to the wide range of applications in which small molecules must be clustered in physical space. Fragment-based drug design[148-150] and the mapping of "hot spots"[151] in protein-protein interfaces provide natural extensions, and Jarvis-Patrick clustering (used in our method) is fast, deterministic, and more accurate than the RMSD-based methods currently in use.

Bridging water molecules are known to be extremely important to the structure, function and dynamics of proteins and nucleic acids. They affect local properties such as hydrogen bonds,[152-157] hydrophobic contacts,[155,156] and electrostatics,[152-154,156] they regulate global properties such as flexibility[154,157-159] and stability,[153-159] and they can contribute directly to catalysis.[152-154,156,157,159] A wealth of data about bridging water interactions is inherently available in explicit-water MD simulations, and PyPAT, a suite of scripts developed as part of this dissertation (Chapter 4 and Appendix 2), provides the first readily available tool for automatically extracting and analyzing this data. With the introduction of PyPAT, we hope investigation of the locations, properties, and dynamics of bridging water molecules will become a standard part of MD studies. In our own work, analysis of bridging water interactions helped to confirm a potential new small-molecule binding site on the surface of DHFR as described in Chapter 3.

Chapter 3 provides new insights into the basic biophysics of DHFR. First, it shows detailed ways in which the folate-binding site pre-organizes to facilitate ligand binding. This information can be used to better understand DHFR-ligand binding and to improve drug design. Second, we find that the DHFR•NADPH complex samples an ensemble of distinct dynamic motions. The makeup and weighting of this ensemble change throughout the catalytic cycle, in a way that is directly analogous to the shifting conformational ensembles. Altered ensembles of conformations and motion between those states are key to the modern understanding of protein binding, enzymatic catalysis, and allosteric control. This idea of ensembles of dynamics is relevant not just to DHFR but to binding and allostery in general. We expect the investigations of dynamic ensembles to take on greater importance wherever protein interactions are studied throughout the field.

Finally, we have discovered a new potential binding site for DHFR. This is particularly important as DHFR is extremely well-studied and has been a major anti-cancer target for over 40 years.[160] It is our hope that thorough investigation of this site could lead to new anti-cancer and anti-microbial drugs. Future research directions include MPS mapping of this site, virtual screening of databases of drug-like compounds, and collaborations for experimental testing.

References

(1) Lehninger, A. L.; Nelson, D. L.; Cox, M. M. *Principles of Biochemistry*; Second ed.; Worth Publishers: New York, NY, 1993.

(2) Böhm, H.-J. In *Protein-Ligand Interactions*; Böhm, H.-J., Schneider, G., Eds. 2005, p 3-20.

(3) Sotriffer, C.; Klebe, G. *Il Farmaco* **2002**, *57*, 243-251.

(4) Ajay; Murcko, M. A. *J. Med. Chem.* **1995**, *38*, 4953-4967.

(5) Jones, S.; Thornton, J. M. *Proc. Natl. Acad. Sci. USA* **1996**, *93*, 13-20.

(6) Ross, P. D.; Subramanian, S. *Biochemistry* **1981**, *20*, 3096-3102.

(7) Dill, K. A.; Bromberg, S. *Molecular driving forces: statistical thermodynamics in chemistry and biology*; Garland Science: New York, NY, 2003.

(8) Kitchen, D. B.; Decornez, H.; Furr, J. R.; Bajorath, J. *Nat. Rev. Drug Discov.* **2004**, *3*, 935-949.

(9) Schneider, H. J. In *Protein-Ligand Interactions*; Böhm, H.-J., Schneider, G., Eds. 2005, p 21-50.

(10) Mancera, R. L. *Curr. Opin. Drug Discovery Dev.* **2007**, *10*, 275.

(11) Leach, A. R. *Molecular Modelling: Principles and Applications*; Prentice Hall, 2001.

(12) Leach, A. R.; Shoichet, B. K.; Peishoff, C. E. *J. Med. Chem.* **2006**, *49*, 5851-5855.

(13) Huang, N.; Jacobson, M. P. *Curr. Opin. Drug Discovery Dev.* **2007**, *10*, 325.

(14) Clark, M.; Guarnieri, F.; Shkurko, I.; Wiseman, J. *J. Chem. Inf. Model.* **2006**, *46*, 231-242.

(15) Efremov, R. G.; Chugunov, A. O.; Pyrkov, T. V.; Priestle, J. P.; Arseniev, A. S.; Jacoby, E. *Curr. Med. Chem.* **2007**, *14*, 393-415.

(16) Pratt, L. R.; Pohorille, A. *Chem. Rev.* **2002**, *102*, 2671-2692.

(17) Dill, K. A.; Truskett, T. M.; Vlachy, V.; Hribar-Lee, B. *Annu. Rev. Biophys. Biomol. Struct.* **2005**, *34*, 173-199.

(18) Sayed, Y.; Wallace, L. A.; Dirr, H. W. *FEBS Lett.* **2000**, *465*, 169-172.

(19) Gschwend, D. A.; Good, A. C.; Kuntz, I. D. *J. Mol. Recognit.* **1996**, *9*, 175-186.

(20) Leatherbarrow, R. J.; Fersht, A. R.; Winter, G. *Proc. Natl. Acad. Sci. USA* **1985**, *82*, 7840-7844.

(21) Haldane, J. B. S. *Enzymes*; Longmans, Green and Co: London, 1930.

(22) Pauling, L. *Chem. Eng. News* **1946**, *24*, 1375-1377.

(23) Keskin, O. *BMC Struct. Biol.* **2007**, *7:31.*

(24) Lindner, A. B.; Eshhar, Z.; Tawfik, D. S. *J. Mol. Biol.* **1999**, *285*, 421-430.

(25) Höfliger, M. M.; Beck-Sickinger, A. G. In *Protein-Ligand Interactions*; Böhm, H.-J., Schneider, G., Eds. 2005, p 107-135.

(26) Koshland, D. E. *Proc. Natl. Acad. Sci. USA* **1958**, *44*, 98-104.

(27) Koshland, D. E. *Angew. Chem. Int. Edit.* **1995**, *33*, 2375-2378.

(28) Kenakin, T. *Trends Pharmacol. Sci.* **1995**, *16*, 188-192.

(29) Ma, B.; Shatsky, M.; Wolfson, H. J.; Nussinov, R. *Protein Sci.* **2002**, *11*, 184-197.

(30) James, L. C.; Tawfik, D. S. *Trends Biochem. Sci.* **2003**, *28*, 361-368.

(31) Monod, J.; Wyman, J.; Changeux, J. P. *J. Mol. Biol.* **1965**, *12*, 88-118.

(32) Koshland, D. E.; Nemethy, G.; Filmer, D. *Biochemistry* **1966**, *5*, 365-385.

(33) Gunasekaran, K.; Ma, B.; Nussinov, R. *Proteins: Struct. Funct. Bioinformatics* **2004**, *57*, 433-443.

(34) Formaneck, M. S.; Ma, L.; Cui, Q. *Proteins: Struct. Funct. Bioinformatics* **2006**, *63*, 846-867.

(35) Swain, J. F.; Gierasch, L. M. *Curr. Opin. Struct. Biol.* **2006**, *16*, 102-108.

(36) Yu, E. W.; Koshland, D. E., Jr. *Proc. Natl. Acad. Sci. USA* **2001**, *98*, 9517-9520.

(37) Ottemann, K. M.; Xiao, W.; Shin, Y.-K.; Koshland, D. E., Jr. *Science* **1999**, *285*, 1751-1754.

(38) Hardy, J. A.; Wells, J. A. *Curr. Opin. Struct. Biol.* **2004**, *14*, 706-715.

(39) Ikeda, Y.; Taniguchi, N.; Noguchi, T. *J. Biol. Chem.* **2000**, *275*, 9150-9156.

(40) Santamaria, B.; Estevez, A. M.; Martinez-Costa, O. H.; Aragon, J. J. *J. Biol. Chem.* **2002**, *277*, 1210-1216.

(41) Goodsell, D. S.; Olson, A. J. *Annu. Rev. Biophys. Biomol. Struct.* **2000**, *29*, 105-153.

(42) Kern, D.; Zuiderweg, E. R. P. *Curr. Opin. Struct. Biol.* **2003**, *13*, 748-757.

(43) Weber, G. *Biochemistry* **1972**, *11*, 864-878.

(44) Gunasekaran, K.; Ma, B.; Ramakrishnan, B.; Qasba, P. K.; Nussinov, R. *Biochemistry* **2003**, *42*, 3674-3687.

(45) Radkiewicz, J. L.; Brooks, C. L., III *J. Am. Chem. Soc* **2000**, *122*, 225-231.

(46) Adams, J.; Johnson, K.; Matthews, R.; Benkovic, S. J. *Biochemistry* **1989**, *28*, 6611-6618.

(47) Adams, J. A.; Fierke, C. A.; Benkovic, S. J. *Biochemistry* **1991**, *30*, 11046-11054.

(48) Ahrweiler, P. M.; Frieden, C. *Biochemistry* **1991**, *30*, 7801-7809.

(49) Cameron, C. E.; Benkovic, S. J. *Biochemistry* **1997**, *36*, 15792-15800.

(50) Chen, J. T.; Taira, K.; Tu, C. P. D.; Benkovic, S. J. *Biochemistry* **1987**, *26*, 4093-4100.

(51) Dion, A.; Linn, C. E.; Bradrick, T. D.; Georghiou, S.; Howell, E. E. *Biochemistry* **1993**, *32*, 3479-3487.

(52) Dion-Schultz, A.; Howell, E. E. *Protein Eng.* **1997**, *10*, 263-272.

(53) Farnum, M. F.; Magde, D.; Howell, E. E.; Hirai, J. T.; Warren, M. S.; Grimsley, J. K.; Kraut, J. *Biochemistry* **1991**, *30*, 11567-11579.

(54) Fierke, C. A.; Benkovic, S. J. *Biochemistry* **1989**, *28*, 478-486.

(55) Howell, E. E.; Booth, C.; Farnum, M.; Kraut, J.; Warren, M. S. *Biochemistry* **1990**, *29*, 8561-8569.

(56) Iwakura, M.; Jones, B. E.; Luo, J.; Matthews, C. R. *J. Biochem. (Tokyo)* **1995**, *117*, 480-488.

(57) Miller, G. P.; Benkovic, S. J. *Biochemistry* **1998**, *37*, 6327-6335.

(58) Miller, G. P.; Benkovic, S. J. *Biochemistry* **1998**, *37*, 6336-6342.

(59) Murphy, D. J.; Benkovic, S. J. *Biochemistry* **1989**, *28*, 3025-3031.

(60) Ohmae, E.; Ishimura, K.; Iwakura, M.; Gekko, K. *J. Biochem. (Tokyo)* **1998**, *123*, 839-846.

(61) Wagner, C. R.; Thillet, J.; Benkovic, S. J. *Biochemistry* **1992**, *31*, 7834-7840.

(62) Cooper, A.; Dryden, D. T. F. *Eur. Biophys. J.* **1984**, *11*, 103-109.

(63) Popovych, N.; Sun, S.; Ebright, R. H.; Kalodimos, C. G. *Nat. Struct. Mol. Biol.* **2006**, *13*, 831-838.

(64) Wand, A. J. *Nat. Struct. Biol.* **2001**, *8*, 926-31.

(65) Homans, S. W. *ChemBioChem* **2005**, *6*, 1585-1591.

(66) Lee, A. L.; Wand, A. J. *Nature* **2001**, *411*, 501-4.

(67) Agarwal, P. K. *Microb. Cell Fact.* **2006**, *5*, 2.

(68) Daniel, R. M.; Dunn, R. V.; Finney, J. L.; Smith, J. C. *Annu. Rev. Biophys. Biomol. Struct.* **2003**, *32*, 69-92.

(69) Palmer, A. G., III *Annu. Rev. Biophys. Biomol. Struct.* **2001**, *30*, 129-155.

(70) Kim, K. S.; Woodward, C. *Biochemistry* **1993**, *32*, 9609-9613.

(71) Fitter, J.; Heberle, J. *Biophys. J.* **2000**, *79*, 1629-1636.

(72) Heller, W. T. *Acta Crystallogr. Sect. D Biol. Crystallogr.* **2005**, *61*, 33-44.

(73) Schramm, V. L.; Shi, W. *Curr. Opin. Struct. Biol.* **2001**, *11*, 657-665.

(74) Moffat, K. *Chem. Rev.* **2001**, *101*, 1569-1582.

(75) Hummer, G.; Schotte, F.; Anfinrud, P. A. *Proc. Natl. Acad. Sci. USA* **2004**, *101*, 15330-15334.

(76) Laberge, M.; Yonetani, T. *IUBMB Life* **2007**, *59*, 528 - 534.

(77) Eisenmesser, E. Z.; Bosco, D. A.; Akke, M.; Kern, D. *Science* **2002**, *295*, 1520-1523.

(78) Osborne, M. J.; Schnell, J.; Benkovic, S. J.; Dyson, H. J.; Wright, P. E. *Biochemistry* **2001**, *40*, 9846–9859.

(79) Eisenmesser, E. Z.; Millet, O.; Labeikovsky, W.; Korzhnev, D. M.; Wolf-Watz, M.; Bosco, D. A.; Skalicky, J. J.; Kay, L. E.; Kern, D.; Contact, N. P. G. *Nature* **2005**, *438*, 117-121.

(80) Kumar, S.; Ma, B.; Tsai, C. J.; Sinha, N.; Nussinov, R. *Protein Sci.* **2000**, *9*, 10-19.

(81) Agarwal, P. K.; Billeter, S. R.; Rajagopalan, P. T. R.; Benkovic, S. J.; Hammes-Schiffer, S. *Proc. Natl. Acad. Sci. USA* **2002**, *99*, 2794-2799.

(82) Hammes, G. G. *Biochemistry* **2002**, *41*, 8221-8228.

(83) Case, D. A.; Darden, T. A.; Cheatham III, T. E.; Simmerling, C. L.; Wang, J.; Duke, R. E.; Luo, R.; Merz, K. M.; Wang, B.; Pearlman, D. A.; Crowley, M.; Brozell, S.; Tsui, V.; Gohlke, H.; Mongan, J.; Hornak, V.; Cui, G.; Beroza, P.; Schafmeister, C.; Caldwell, J. W.; Ross, W. S.; Kollman, P. A. **2004**, University of California, San Francisco, AMBER 8.

(84) Cornell, W. D.; Cieplak, P.; Bayly, C. I.; Gould, I. R.; Merz, K. M.; Ferguson, D. M.; Spellmeyer, D. C.; Fox, T.; Caldwell, J. W.; Kollman, P. A. *J. Am. Chem. Soc.* **1995**, *117*, 5179-5197.

(85) Brooks, B. R.; Bruccoleri, R. E.; Olafson, B. D.; States, D. J.; Swaminathan, S.; Karplus, M. *J. Comput. Chem* **1983**, *4*, 187-217.

(86) Jorgensen, W. L.; Tirado-Rives, J. *J. Am. Chem. Soc.* **1988**, *110*, 1657-1666.

(87) Shaw, D. E.; Deneroff, M. M.; Dror, R. O.; Kuskin, J. S.; Larson, R. H.; Salmon, J. K.; Young, C.; Batson, B.; Bowers, K. J.; Chao, J. C.; Eastwood, M. P.; Gagliardo, J.; Grossman, J. P.; Ho, C. R.; Ierardi, D. J.; Kolossvary, I.; Klepeis, J.; Layman, T.; McLeavey, C.; Moraes, M. A.; Mueller, R.; Priest, E. C.; Shan, Y.; Spengler, J.; Theobald, M.; Towles, B.; Wang, S. C. *Proceedings of the 34th annual international conference on Computer architecture* **2007**, 1-12.

(88) Duan, Y.; Kollman, P. A. *Science* **1998**, *282*, 740.

(89) Seibert, M. M.; Patriksson, A.; Hess, B.; van der Spoel, D. *J. Mol. Biol.* **2005**, *354*, 173-183.

(90) Borrell, B. *Nature* **2008**, *451*, 240-243.

(91) Sawaya, M. R.; Kraut, J. *Biochemistry* **1997**, *36*, 586-603.

(92) Meagher, K. L.; Carlson, H. A. *J. Am. Chem. Soc* **2004**, *126*, 13276-13281.

(93) Lerner, M. G.; Bowman, A. L.; Carlson, H. A. *J. Chem. Inf. Model.* **2007**, *47*, 2358-2365.

(94) Mu, Y.; Stock, G. *Biophys. J.* **2006**, *90*, 391-399.

(95) Ma, J. *Structure* **2005**, *13*, 373-380.

(96) Brooks, B.; Karplus, M. *Proc. Natl. Acad. Sci. USA* **1983**, *80*, 6571-6575.

(97) Amadei, A.; Linssen, A. B. M.; Berendsen, H. J. C. *Proteins: Struct., Funct., Genet.* **1993**, *17*, 412–425.

(98) Aalten, D.; Groot, B. L. D.; Findlay, J. B. C.; Berendsen, H. J. C.; Amadei, A. *Journal of Computational Chemistry* **1997**, *18*, 169-181.

(99) Mongan, J. *J. Comput-Aided. Mol. Des.* **2004**, *18*, 433-436.

(100) Cauchy, A. L. *Cours d'analyse de l'École Royale Polytechnique, 1ère partie: Analyse algébrique.* Paris: p. 373, 1821.

(101) Weisstein, E. W. In *MathWorld--A Wolfram Web Resource*. 2008. http://mathworld.wolfram.com/CauchysInequality.html

(102) Weisstein, E. W. In *MathWorld--A Wolfram Web Resource*. 2008. http://mathworld.wolfram.com/StatisticalCorrelation.html

(103) Ichiye, T.; Karplus, M. *Proteins: Struct., Funct., Genet.* **1991**, *11*, 205-217.

(104) Rod, T. H.; Radkiewicz, J. L.; Brooks, C. L., III *Proc. Natl. Acad. Sci. USA* **2003**, *100*, 6980-6985.

(105) Li, L.; Uversky, V. N.; Dunker, A. K.; Meroueh, S. O. *J. Am. Chem. Soc.* **2007**, *129*, 15668-15676.

(106) Eckhardt, R. *Los Alamos Sci.* **1987**, *15*, 131-143.

(107) Metropolis, N. *Los Alamos Sci.* **1987**, *15*, 125-130.

(108) Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; Teller, E. *J. Chem. Phys.* **1953**, *21*, 1087-1092.

(109) Carlson, H. A.; Masukawa, K. M.; Rubins, K.; Bushman, F. D.; Jorgensen, W. L.; Lins, R. D.; Briggs, J. M.; McCammon, J. A. *J. Med. Chem* **2000**, *43*, 2100-2114.

(110) Jorgensen, W. L. **2000**, Yale University, New Haven, CT, BOSS 4.2.

(111) Li, Z.; Scheraga, H. A. *Proc. Natl. Acad. Sci. USA* **1987**, *84*, 6611-6615.

(112) Ming, D.; Wall, M. E. *Proteins: Struct. Funct. Bioinformatics* **2005**, *59*, 697-707.

(113) Kullback, S.; Leibler, R. A. *Ann. Math. Stats.* **1951**, *22*, 79-86.

(114) Ota, N.; Agard, D. A. *J. Mol. Biol* **2005**, *351*, 345-354.

(115) Carlson, H. A. *Curr. Opin. Chem. Biol.* **2002**, *6*, 447-452.

(116) Lyne, P. D. *Drug Discov. Today* **2002**, *7*, 1047-1055.

(117) Pickett, S. In *Protein-Ligand Interactions*; Böhm, H.-J., Schneider, G., Eds. 2005, p 73-105.

(118) Teodoro, M. M. L.; Kavraki, L. L. E. *Curr. Pharm. Des.* **2003**, *9*, 1635-48.

(119) Subramanian, J.; Sharma, S.; Rao, C. B. *ChemMedChem* **2007**, *in press*.

(120) Damm, K. L.; Carlson, H. A. *J. Am. Chem. Soc.* **2007**, *129*, 8225-8235.

(121) Meagher, K. L.; Lerner, M. G.; Carlson, H. A. *J. Med. Chem* **2006**, *49*, 3478-3484.

(122) Bowman, A. L.; Lerner, M. G.; Carlson, H. A. *J. Am. Chem. Soc.* **2007**, *129*, 3634-3640.
(123) Bowman, A. L.; Nikolovska-Coleska, Z.; Zhong, H.; Wang, S.; Carlson, H. A. *J. Am. Chem. Soc.* **2007**, *129*, 12809-12814.
(124) Voet, D.; Voet, J. *Biochemistry*; 2nd ed.; John Wiley & Sons: New York, NY, 1995.
(125) Voet, D.; Voet, J. In *Biochemistry*; 2nd ed. ed.; John Wiley & Sons, Inc.: New York, NY, 1995, p 762.
(126) Schweitzer, B. I.; Dicker, A. P.; Bertino, J. R. *FASEB J.* **1990**, *4*, 2441-2452.
(127) Berg, J. M.; Tymoczko, J. L.; Stryer, L. *Biochemistry*; 5th ed.; W. H. Freeman and Company: New York, 2002.
(128) Schnell, J. R.; Dyson, H. J.; Wright, P. E. *Annu. Rev. Biophys. Biomol. Struct.* **2004**, *33*, 119-140.
(129) Agarwal, P. K.; Billeter, S. R.; Hammes-Schiffer, S. *J. Phys. Chem. B* **2002**, *106*, 3283-3293.
(130) Feeney, J. *Angew. Chem. Int. Edit.* **2000**, *39*, 290-416.
(131) Osborne, M. J.; Schnell, J.; Benkovic, S. J.; Dyson, H. J.; Wright, P. E. *Biochemistry* **2001**, *40*, 9846-9859.
(132) Boehr, D. D.; McElheny, D.; Dyson, H. J.; Wright, P. E. *Science* **2006**, *313*, 1638-1642.
(133) Hammes-Schiffer, S.; Benkovic, S. J. *Annu. Rev. Biochem.* **2006**, *75*, 519-541.
(134) Venkitakrishnan, R. P.; Zaborowski, E.; McElheny, D.; Benkovic, S. J.; Dyson, H. J.; Wright, P. E. *Biochemistry* **2004**, *43*, 16046-16055.
(135) Khavrutskii, I. V.; Price, D. J.; Lee, J.; Brooks, C. L., III *Protein Sci.* **2007**, *16*, 1087-1100.
(136) McElheny, D.; Schnell, J. R.; Lansing, J. C.; Dyson, H. J.; Wright, P. E. *Proc. Natl. Acad. Sci. USA* **2005**, *102*, 5032-5037.
(137) Thorpe, I. F.; Brooks, C. L., III *Proteins: Struct. Funct. Bioinformatics* **2004**, *57*, 444-457.
(138) Rod, T. H.; Brooks, C. L., III *J. Am. Chem. Soc.* **2003**, *125*, 8718-8719.
(139) Fierke, C. A.; Johnson, K. A.; Benkovic, S. J. *Biochemistry* **1987**, *26*, 4085-4092.
(140) Duda, R. O.; Hart, P. E.; Stork, D. G. *Pattern Classification*; Wiley-Interscience, 2000.
(141) Kauffman, L.; Rousseeuw, P. J. *Finding Groups in Data: An Introduction to Cluster Analysis*; John Wiley & Sons, Inc.: New York, 1990.
(142) Jarvis, R. A.; Patrick, E. A. *IEEE T. Comput.* **1973**, *22*, 1025-1034.
(143) Boehr, D. D.; McElheny, D.; Dyson, H. J.; Wright, P. E. *Science* **2006**, *313*, 1638.
(144) Evensen, E.; Joseph-McCarthy, D.; Karplus, M. **1997**, Harvard University, Cambridge, MA, USA, MCSS version 2.1.
(145) Böhm, H. J. *J. Comput-Aided. Mol. Des.* **1992**, *6*, 61-78.
(146) Schechner, M.; Sirockin, F.; Stote, R. H.; Dejaegere, A. P. *J. Med. Chem.* **2004**, *47*, 4373-4390.
(147) Sirockin, F.; Sich, C.; Improta, S.; Schaefer, M.; Saudek, V.; Froloff, N.; Karplus, M.; Dejaegere, A. *J. Am. Chem. Soc.* **2002**, *124*, 11073-11084.
(148) Carr, R. A. E.; Congreve, M.; Murray, C. W.; Rees, D. C. *Drug Discov. Today* **2005**, *10*, 987-992.

(149) Erlanson, D. A.; McDowell, R. S.; O'Brien, T. *J. Med. Chem.* **2004**, *47*, 3463-3482.
(150) Rees, D. C.; Congreve, M.; Murray, C. W.; Carr, R. *Nat. Rev. Drug Discov.* **2004**, *3*, 660-672.
(151) Vajda, S.; Guarnieri, F. *Curr. Opin. Drug Discovery Dev.* **2006**, *9*, 354.
(152) Bergqvist, S.; Williams, M. A.; O'Brien, R.; Ladbury, J. E. *J. Mol. Biol.* **2004**, *336*, 829-842.
(153) Jayaram, B.; Jain, T. *Annu. Rev. Biophys. Biomol. Struct.* **2004**, *33*, 343-361.
(154) Nakasako, M. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* **2004**, *359*, 1191-1206.
(155) Plumridge, T. H.; Waigh, R. D. *J. Pharm. Pharmacol.* **2002**, *54*, 1155-1179.
(156) Saenger, W. *Annu. Rev. Biophys. Bio.* **1987**, *16*, 93-114.
(157) Westhof, E. *Annu. Rev. Biophys. Bio.* **1988**, *17*, 125-144.
(158) Feig, M.; Pettitt, B. M. *Structure* **1998**, *6*, 1351-4.
(159) Smith, J. C.; Merzel, F.; Bondar, A. N.; Tournier, A.; Fischer, S. *Philosophical Transactions: Biological Sciences* **2004**, *359*, 1181-1190.
(160) Huennekens, F. M. *Protein Sci.* **1996**, *5*, 1201.

Chapter 2

Incorporating dynamics in *E. coli* dihydrofolate reductase enhances structure-based drug discovery

Introduction

Dihydrofolate reductase (DHFR; EC. 1.5.1.3) catalyzes the reduction of dihydrofolate (DHF) to tetrahydrofolate (THF) (and more slowly, folate to dihydrofolate) while concurrently oxidizing the nicotinamide adenine dinucleotide phosphate (NADPH) cofactor.[1] DHFR is a small enzyme that plays an essential role in various cellular processes including the biosynthesis of DNA.[2] It is the only source of THF, an essential precursor for purine and thymidylate biosynthesis, and hence, it is a long-standing target for anticancer drugs and antibacterial agents. Moreover, *Escherichia coli* DHFR (ecDHFR) is a canonical system for studying enzyme structure, dynamics, and catalysis. In fact, ecDHFR was recently the focus of a data-mining and docking competition[3] which highlighted possible improvements needed in the field.[4]

Protein flexibility and dynamics are of utmost importance in understanding the structure and mechanism of DHFR. The role of protein dynamics in DHFR catalysis has been well studied both computationally[5-7] and experimentally through NMR spectroscopy[8-10] and multiple X-ray crystallography structures.[11] The M20 loop of ecDHFR (residues 9 – 24) adopts three different conformations: closed, open, and occluded.[11] When the substrate and cofactor are both bound, the closed conformation is adopted, in which the M20 loop is packed against the nicotinamide ring of the cofactor[11]. The closed form presents the only conformation in which the substrate and cofactor are arranged favorably for reaction, and it is also present in the preceding state of the catalytic cycle, DHFR·NADPH.[6] The closed form is also the only conformation seen in DHFR from all other species, regardless of the crystal packing and ligands bound in the substrate and cofactor binding sites.[11] The occluded M20 loop is observed in the product complexes (DHFR·NADP$^+$·THF, DHFR·THF, and DHFR·NADPH·THF). It is an unproductive conformation where the central part of the M20 loop forms a helix, blocking access to the binding site for the nicotinamide moiety of the cofactor. This forces the nicotinamide of the cofactor out into solvent, making it unresolved in the crystal structures of DHFR·NADP$^+$·THF and DHFR·NADPH·THF.[11] The open loop is a conformational intermediate between the extremes of the closed and occluded loops. The significance of M20 loop dynamics in ligand binding and catalysis has been an area of great interest, driven by the persuasive evidence of conformational change in the loop during the catalytic cycle and its interaction with the substrate and cofactor.[11]

Furthermore, there is evidence that the degree and pattern of protein flexibility changes throughout the catalytic cycle.[11] All regions of DHFR dynamically contribute to substrate binding and chemical turnover.[12] The M20 loop conformations and sub-domain rotations are both pivotal during catalysis, implying coupling to the reaction coordinate.[11,13] The M20 loop undergoes fluctuations on the ps – ns timescale, some of which are analogous to the rate-limiting step of product release.[14] During molecular dynamics (MD) simulations of the DHFR·NADPH·DHF complex, strongly correlated and anticorrelated motions were observed across distant regions of the protein.[5] This extensive coupling was only seen in the reactant form, not in either the DHFR·NADP$^+$·THF or DHFR·NADPH·THF complexes, implying that these motions may be necessary for catalysis.[5] NMR relaxation experiments indicate that binding events influence not only active-site dynamics but also distal motions.[9,10]

In light of this evidence, it is imperative that protein flexibility be included when considering inhibitor binding in ecDHFR. Here, we present a computational study of the influence of conformational behavior of ecDHFR on inhibitor identification. We employ multiple protein structures (MPS) to incorporate protein flexibility in structure-based drug discovery. Our technique identifies complementary interactions within the binding site of an ensemble of conformational states.[15,16] Conserved regions, where the same complementary interactions are consistently made in a majority of the MPS, define the most essential binding hotspots for different chemical functionalitites. The conformational states can be taken from MD simulation,[16-18] X-ray crystallographic structures[15,19,20] or an NMR ensemble[20]. The method, initially applied to HIV integrase,[15,16] has been successfully extended to HIV-1 protease,[17,18] and other species of DHFR.[19]

The dynamic motions of ecDHFR are crucial to ligand binding and subsequent catalysis. Additionally, incorporating protein flexibility expands the chemical and conformational space of the predicted ligands. The use of pharmacophore models, rather than specific, predetermined chemical scaffolds, also greatly increases the available chemical space for the hit list. The incorporation of greater flexibility through extending the length of the sampled MD simulation has shown an improvement of the identification of known HIV-1 protease inhibitors over drug-like non inhibitors with our method.[17,18]

Our recent work with DHFR showed that the inclusion of protein flexibility did not result in a loss of specificity across DHFR from different species.[19] In this study, we have used snapshots from an MD simulation to develop models from the unliganded complex DHFR·NADPH with the M20 loop in both the closed and open conformations. This use of the DHFR·NADPH complex is analogous to that of using *apo* structures; the substrate binding site has no predetermined conformation relating to a specific inhibitor. We also utilize experimental structures to develop a pharmacophore model, in a similar way to our previous work based on DHFR from other species.[19] The MPS method has already been shown to be successful when investigating *apo* HIV-1 protease structures[17,18] and the ecDHFR pharmacophore models developed here are highly selective for known DHFR inhibitors over drug-like non-inhibitors. We also show that extending the length of the MD simulation to increase sampling of protein flexibility improves model performance, especially in the model resulting from the simulation of DHFR with the M20 loop in the open, unproductive orientation. This study is a further indication of how important the inclusion of conformational behavior can be in enhancing drug discovery.

Methods

Protein preparation. The starting models for the MD simulations were based on two crystal structures of the wild-type *E. coli* DHFR·NADPH complex, representing both the closed (1RX1) and open (1RA1) M20 loop.[11] The 1RA1 structure was altered to include the entire cofactor, building in the nicotinamide ribose group (Fig. 1) by direct alignment with the coordinates in the 1RX1 structure. In complexes with the open loop, the nicotinamide ribose group is mobile and moves in and out of the cofactor binding site. We chose to represent it in the binding pocket, as electron density indicates that the pocket is occupied 75% of the time.[11] The occupancy is increased to 100% upon binding of the substrate and closure of the M20 loop.[11] This conformation is more suited to modeling inhibitor binding and hence more appropriate for this study.

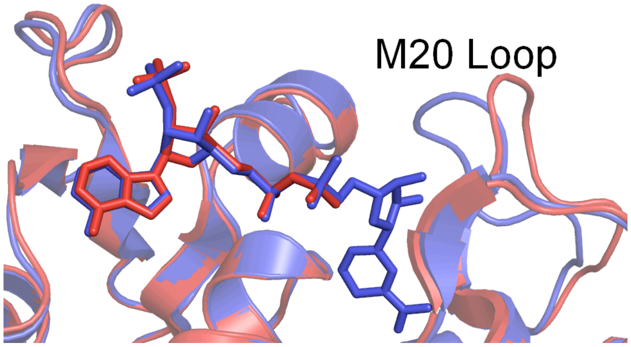


Figure 1. The crystal structures of 1RX1[11] in blue with the closed M20 loop conformation and 1RA1[11] in red in the open M20 loop conformation. NADPH is shown in stick representation; the nicotinamide ribose moiety of the cofactor is not resolved in the 1RA1 structure and was built in using the appropriate coordinates from the 1RX1 structure. The M20 loop is to the right of the nicotinamide.

Side-chain orientations and protonation states were checked with a custom script in PyMOL.[21] Hydrogen atoms were placed using the LEaP routine[22] and optimized with conjugate gradient energy minimization (convergence criterion: RMS gradient <0.0001 kcal/mol Å). The force-field parameters for the cofactor were provided by Ryde *et al*.[23] The protein was then solvated in a truncated octahedral box of TIP3P[24] water extending 15 Å from the surface of the protein. To give the system a net neutral charge, 15 sodium counter-ions were added 10 Å from the protein surface. A preliminary minimization of the entire system was performed with 300 steps of conjugate gradient method to remove initial bad contacts. The proteins heavy atoms were then held fixed. The water molecules were heated from 10 K to 310 K over 50 ps, followed by equilibration (310 K) for 50 ps at constant volume and a subsequent 300 ps with constant pressure. The entire system then underwent heating from 10 K to 310 K over 50 ps, followed by equilibration (310 K) for 50 ps at constant volume. Finally, 500 ps of all-atom equilibration at constant pressure was performed, and 5 ns of production phase MD were collected. Analysis of the MD (Fig. 2) trajectory showed that the system takes an unusually long time to equilibrate, particularly the more mobile open conformation. This same slow equilibration was also found in other dynamics studies of ecDHFR.[5] For this reason, we discarded the first nanosecond of production dynamics, leaving four nanoseconds of sampling dynamics. This was done for both simulations to maintain consistency. Snapshots were saved from the dynamics trajectory every 100 ps. All simulations were carried out with AMBER.[25]

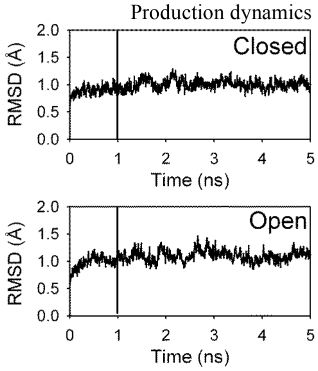


Figure 2. Plots reflecting the RMSD between the equilibration structure of (a) M20 closed-loop and (b) M20 open-loop ecDHFR and respective trajectory snapshots versus time.

Comparison to pharmacophore models from X-ray crystallographic structures. In our previous work, we utilized X-ray crystallographic structures of DHFR from other species which had seven or more complexes in the PDB.[19] There are only crystal structures of wild-type ecDHFR bound to four unique ligands,[11,26] but it is important to compare models based on MD to those based on experimental structures. X-ray crystallographic structures of the wild-type ecDHFR with unique ligands bound and in which the cofactor was fully resolved were downloaded from the PDB.[27] If a structure with an identical ligand existed, the structure with the better resolution was taken. The four resultant structures (1RA2,[11] 1RC4,[11] 1RX3,[11] and 2ANQ[26]) had an average Cα RMSD of 0.56 Å. Fig. 3 shows that the flexibility comes almost entirely in the M20 loop region; there is very little conformational variation in the rest of the structure.

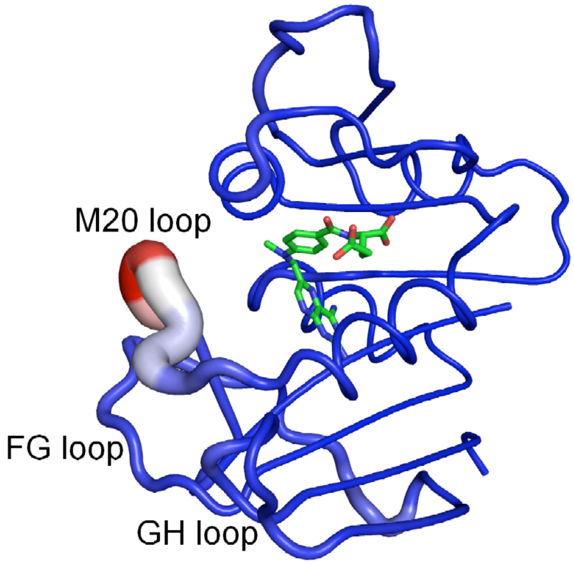


Figure 3. Average backbone structure for four X-ray crystallographic structures of ecDHFR. A red, thicker tube indicates greater RMSD across the ensemble, whereas a blue, narrow tube shows limited flexibility. Loop regions are labeled in the model. As expected, greater flexibility is seen in the loops than in the core of the protein. The bound conformation of methotrexate is superimposed on the model to orient the reader.

Probe flooding, minimization, and clustering. The procedure for the model set-up was similar to that for our recent work on DHFR from other species.[19] The active site of each snapshot was flooded with 1000 small molecule probes using an 11 Å sphere centered at the mid-point of Phe31 and the nicotinamide ring of the cofactor, using an in-house program to randomly pack the probes in their initial locations. The size and shape of the binding site (in particular, the fact that it is relatively small and deep) required a denser placement, in order to fully explore relevant interactions, as compared to our earlier studies on HIV protease. This was achieved by doubling the number of probes used, while keeping the flooding sphere size similar. Benzene probes were used to identify aromatic and hydrophobic interactions, ethane probes were used to distinguish hydrophobic interactions from aromatic, and methanol probes were used to identify hydrogen-bonding interactions. Low-temperature Monte Carlo minimizations were performed by the Multi-Unit Search for Interacting Conformers (MUSIC) routine in BOSS,[28] using the OPLS force field.[29] In MUSIC, the protein is held fixed while the probe molecules undergo simultaneous multiple gas-phase minimization. The probes cluster into local minima which define complementary binding regions. Clusters containing eight or more probes are identified with an automated procedure, based on Jarvis-Patrick methodology (See Chapter 5). Each cluster is represented by its "parent", the lowest-energy probe in the group.

Each snapshot was overlaid to the final equilibration structure using a Gaussian-weighted RMSD alignment[30] to give a common frame of reference. Parent probes within 8 Å of the center of the binding site (defined by Phe31 Cγ) were combined and clustered to give "consensus clusters". A consensus cluster must contain parent probes from ≥ 50 % of the protein conformations. Pharmacophore elements were centered on the average position of the probes in each consensus cluster, and the radius was given by the RMSD of the probes in the cluster to the center of the cluster. Protein flexibility is implicitly included because only probes in conserved (rigid) regions are incorporated into consensus clusters. In flexible regions, where no consensus clusters were present, no limitations or requirements are set. This expands the range of chemical and conformational space that can be explored. Small areas of highly conserved steric constraints within the substrate binding pocket were represented by the inclusion of four excluded volume elements. These were each given a radius of 1.5 Å and were centered at the average position of Ile5 C, Ala 7 Cα, Phe31 Cγ, and Ile50 Cα.

Pharmacophore models were created using snapshots from the first one, two, and the full four nanoseconds of sampling dynamics. A total of 10 snapshots were used for the one nanosecond model (taken every 100 ps), and 20 snapshots were used for the two and four nanosecond models (taken every 100 ps or 200 ps, respectively).

Creation of the ligand databases. MOE[31] was used to screen three databases against the models: one set of 50 high-affinity inhibitors, one set containing 541 general DHFR inhibitors, and a general set of drug-like non-inhibitors. A total of 107 ecDHFR inhibitors were taken from the literature, each with an $IC_{50} \leq 1$ μM. The full set of structures and affinity data are provided with references in the supplemental information (the dataset is completely compatible with that from our earlier work). These inhibitors were merged with the database of 493 DHFR inhibitors from our previous work[19] to yield a database of 591 unique inhibitors (there were nine duplicates). From this database, the top 50 ecDHFR inhibitors were selected for the high-affinity dataset; the IC_{50} values ranged from 2 – 28 nM. The remainder of 541 inhibitors was used as the general DHFR inhibitor set. Our previous set of 2303 drug-like decoy molecules,[19] obtained from the CMC,[32] was also used in this work. These decoy molecules had a molecular weight great than 100 and contained one hydrogen-bond donor and one aromatic atom, the set did not contain any folate-like molecules.[19] Rule-based torsion driving in OMEGA[33] was used to produce multiple conformations of each molecule, using an energy cutoff of 14 kcal/mol calculated with the MMFF force field and a heavy-atom RMSD criterion of 1 Å. These pre-generated conformations were compared to the pharmacophore models. A compound was counted as a hit if one conformation could be aligned to the pharmacophore coordinates.

Evaluation of pharmacophore models. The predictive performance of each pharmacophore model was estimated by generating a receiver operator characteristic (ROC) curve, where the percentage of true inhibitors (true positives) found is plotted against the percentage of drug-like non-inhibitors (false positives) identified. The optimal model would lie at the point (0,100) identifying no false positives but finding all true positives, whereas models lying on a line that passes through the origin with a slope of 1 would be no better than random.

As in our previous studies, the number of elements required was varied, *e.g.* requiring n, $n - 1$ or $n - 2$ elements from an n-site pharmacophore model.[19] A multiplication factor of 1 to 3 in ⅓ increments was used to increase the radius of each pharmacophore element (*e.g.* radii = 1×RMSD, 1.3×RMSD, 1.7×RMSD). This is in keeping with similar studies on HIV-1 protease.[17,18] However, the much smaller RMSD values of the consensus clusters found in the X-ray model required extension to 6 × RMSD to produce reasonable models with radii in the range 0.8 – 4.1 Å (in our previous work we found optimal models had elements with radii in the range 0.6 – 4.1 Å).[19]

Results and discussion

Comparison of MPS models based on the closed-loop conformations. Pharmacophore models resulting from the 1 and 2 ns MD simulations of ecDHFR in the closed-loop conformation have six elements, while the model from the longer 4 ns of sampling has just five. Fig. 4 shows that there are five common elements between these three models. An aromatic element (aromatic/hydrophobic in the 1-ns model) makes an interaction with Phe31 at the bottom of the binding site. Two hydrogen-bond donor elements make interactions with NADPH O7 and Asp27 Oδ2 deep in the pocket. The pteridine type moiety of folate and known inhibitors is characteristically mapped by these three elements.

At a region corresponding to the *p*-aminobenzoate linker region of the substrate an aromatic (aromatic/hydrophobic in the 1-ns and 2-ns models) element is present, making an interaction with the cofactor. A hydrogen-bond acceptor in the mouth of the binding site makes an interaction with Arg57; this contact can be fulfilled by a moiety such as the glutamate tail of folate and some known inhibitors such as methotrexate. The 1-ns and 2-ns models also contain an additional sixth element in this area, an aromatic/hydrophobic site, which is lost with the greater flexibility incorporated in the 4-ns model. This sixth element occupies the Cα region of the glutamate tail of folate; however, it is not essential to high-affinity inhibitors of ecDHFR. It is reasonable that incorporating more flexibility shows this site to be non-essential for inhibitors.

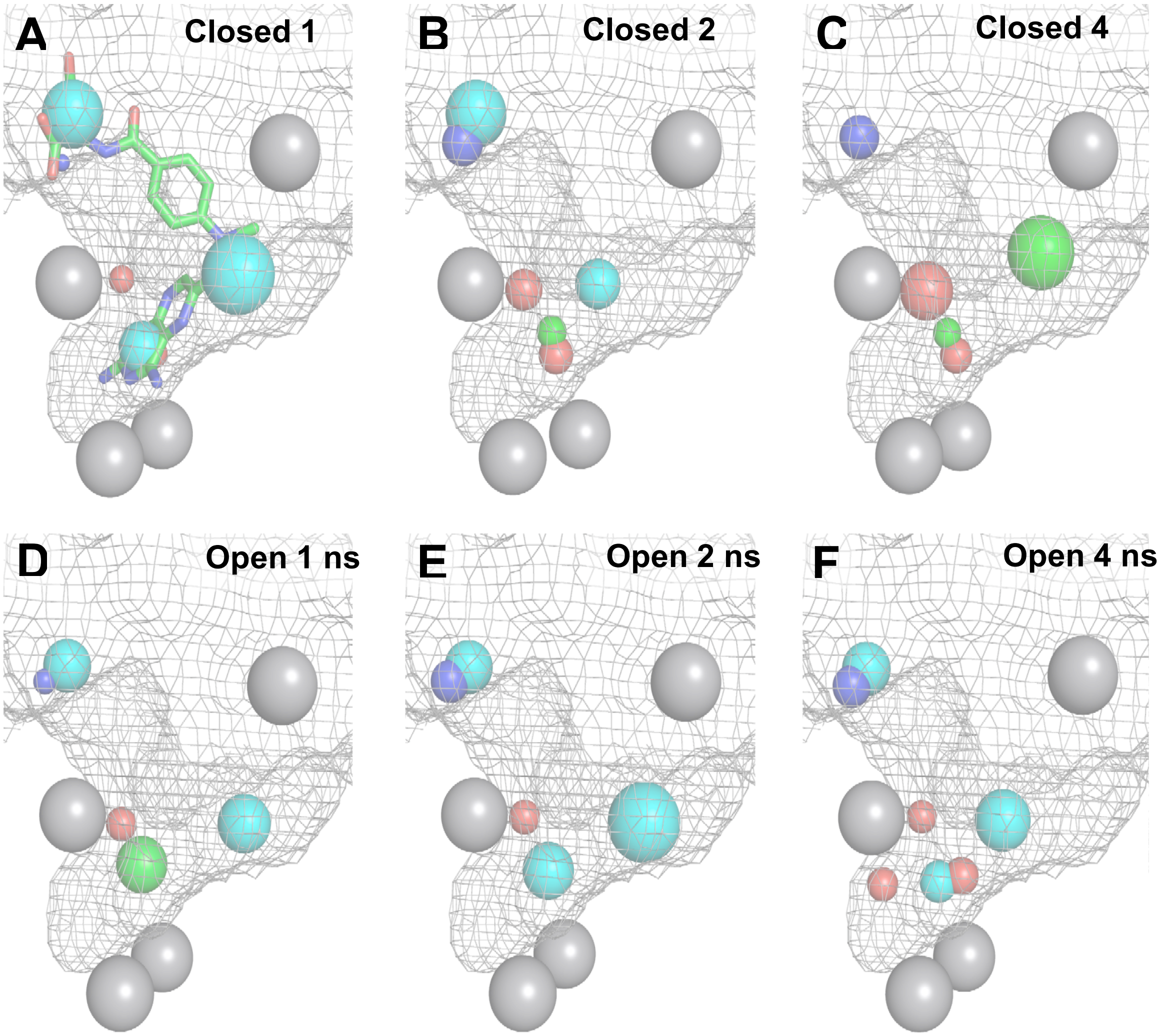


Figure 4. Pharmacophore models are given based on snapshots from (a) 1 ns, (b) 2 ns, and (c) 4 ns of sampling dynamics of the M20 closed-loop conformation of *E. coli* DHFR·NADPH (based on 1RX1). Also given are the models resulting from (d) 1 ns, (e) 2 ns, and (f) 4 ns sampling dynamics of the M20 open-loop conformation of *E. coli* DHFR·NADPH (based on 1RA1). Elements are shown with radii = 1×RMSD: green spheres map aromatic interactions, cyan spheres require aromatic or hydrophobic groups, blue are hydrogen-bond acceptors, and red are hydrogen-bond donors. The gray spheres are excluded volumes. The molecular surface, from 1RX1 or 1RA1 appropriately, is shown in gray; the bound conformation of methotrexate is superimposed on the 1-ns model to orient readers familiar with the DHFR binding site.

Comparison of MPS models based on the open-loop conformations. The models from the 1-ns and 2-ns simulations of the open M20 loop both had 5 pharmacophore elements: the aromatic/hydrophobic element at the bottom of the pocket (aromatic in the 1-ns model); the hydrogen-bond donor to Asp27 Oδ2; the aromatic/hydrophobic element midway in the active site; a hydrogen-bond acceptor in the mouth of the binding pocket; and an aromatic/hydrophobic element, also in the mouth of the pocket. The 4-ns model has two additional hydrogen-bond donor sites, one similar to that seen in the closed-loop interacting with NADPH O7 and another interacting with Asp27 Oδ1. The appearance of these sites adds more specificity, which improves the model by reducing the percentage of false positives identified (see performance discussion below). In particular, the appearance of hydrogen-bond donor site with relation to the co-factor is seems to be crucial to the model's performance. In the closed loop simulation, the distance between the Cδ of Phe21 and O7 of the cofactor averages 7.8 Å. When the M20 loop is open, NADPH is not packed in close to the substrate binding site,[11] and this same distance is longer. In the open-loop simulations, the distance between Phe31 Cδ and the co-factor O7 averages 8.3 Å over the first two nanoseconds, but it is 7.9 Å in the last two nanoseconds. As more dynamics are incorporated into the pharmacophore model, conformations of the loop more similar to the closed form are sampled, which is consistent with Wright's observation that small populations of alternate conformations are part of the observerable ensemble.[9] This may explain why the pharmacophore element representing an interaction with NADPH O7 is only present in the 4-ns pharmacophore model based on the open-loop simulation.

Performance of the MPS pharmacophore models. Each model was screened against three databases: one set of 50 potent ecDHFR inhibitors, one set of 541 general DHFR inhibitors, and one broad set of 2303 drug-like non-inhibitors. The number of elements required was set by allowing a partial match in the pharmacophore search. The radius of each element was manually altered in the pharmacophore query editor in MOE[31] to give the required radii multiplication *e.g.* 2×RMSD. The best pharmacophore models hits the maximum of true positives with the least number of false positives being identified; the optimal model is defined as having the shortest distance from (0,100). The performance of pharmacophore models generated from 1, 2, and 4 ns of dynamic sampling of DHFR·NADPH with both the closed (1RX1) and open (1RA1) M20 loop conformations is given in Fig. 5. Both sets of models identify DHFR inhibitors, both species-specific and general, over non-inhibitors. The optimal pharmacophore model for each system preferentially hits potent ecDHFR inhibitors over other general DHFR inhibitors. It is a concern that incorporating protein flexibility to expand the chemical space may lead to a more general model with a lack of specificity, but this is not the case with the MPS method.[19]

It is encouraging that using structures with cofactor but no folate mimic, analogous to *apo* structures and hence not possessing conformational bias towards bound inhibitors, produced effective pharmacophore models. Our MPS method is one of very few that can successfully identify inhibitors for unbound, open binding sites. However, it is clear that the models for the closed-loop, the conformation most similar to that of the bound complex prior to reaction, are superior to those of the open loop.

The optimal pharmacophore model from the closed-loop 1-ns simulation (6 from 6, radii = 2.3 × RMSD) identified 86% of the high-affinity ecDHFR inhibitors and 56% of the general DHFR inhibitors with just 6% of the false positives being hit. The optimal 2-ns model (also 6 from 6, radii = 2.3 × RMSD) also identified 86% of the high-affinity inhibitors, a higher percentage (61%) of the less potent DHFR inhibitors were also identified, and even fewer of the drug-like non-inhibitors were hit (4%). The optimal closed-loop 4-ns model (4 from 5, radii = 1.7 × RMSD) results in an increase of both the high-affinity ecDHFR (88%) and general DHFR (84%) inhibitors identified while maintaining a low false positive hit rate (11%). The bias towards high-affinity ecDHFR inhibitors over other DHFR inhibitors indicates that specificity has not been lost during the incorporation of protein flexibility. In fact, more flexibility results in the identification of more known inhibitors.

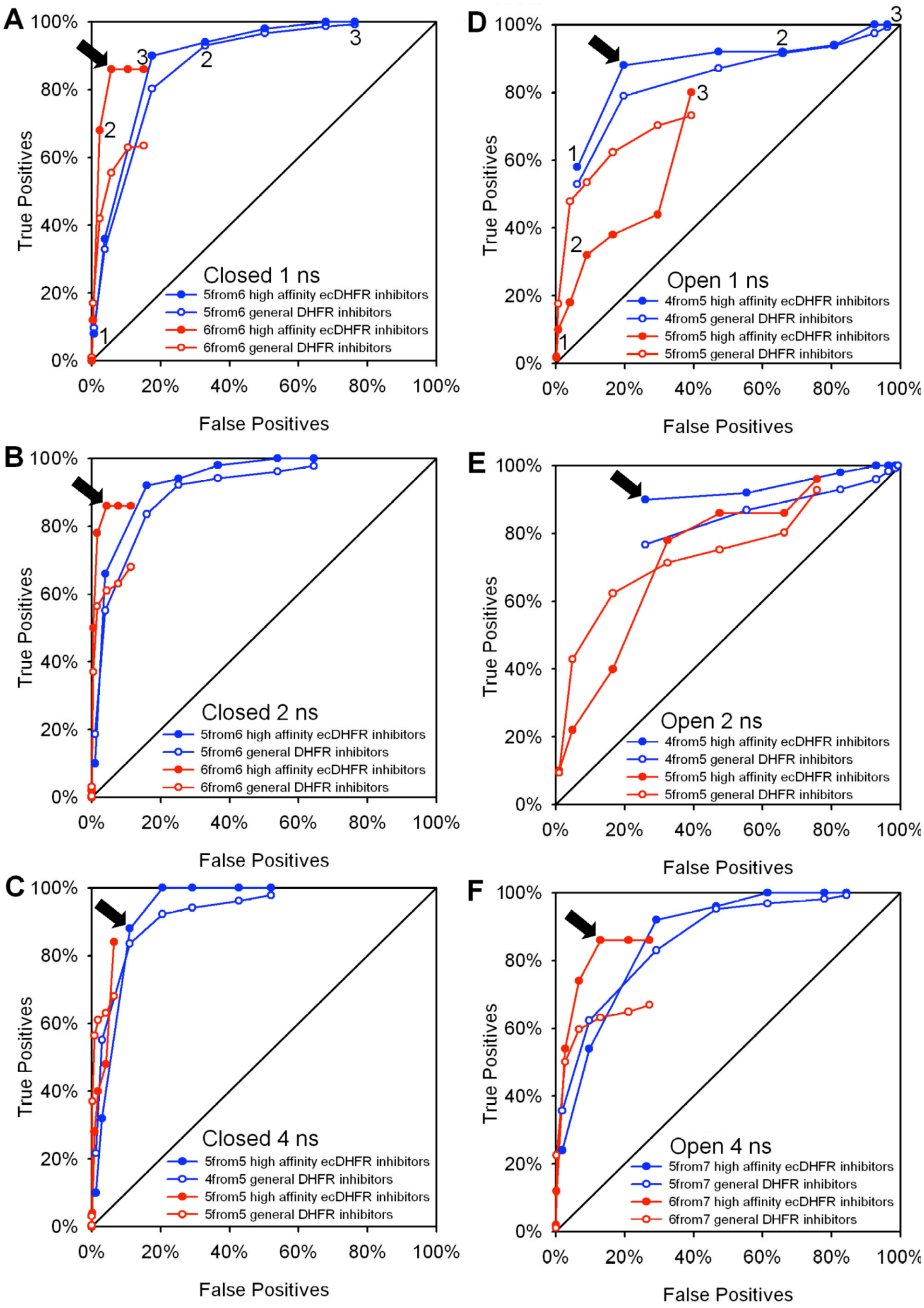


Figure 5. ROC curves for the closed-loop and open-loop pharmacophore models. Series with filled data points are results from screening the high-affinity ecDHFR database; those with open data points are the screening of general DHFR inhibitors. Shown in red are results from models requiring *n* hits from an *n*-site pharmacophore model, shown in blue are results from models requiring *n* – 1 hits from an *n*-site pharmacophore model. Points along each series represent an increase in pharmacophore element radii of 1×RMSD to 3×RMSD. Arrows indicate the optimal model.

Although the optimal pharmacophore model from the open-loop 1-ns simulation (4 from 5, radii = 1.3 × RMSD) identifies 88% of potent ecDHFR inhibitors and 79% of general DHFR inhibitors, it also hits 20% of the false positives. The optimal model from the 2-ns simulation (4 from 5, radii = 1.0 × RMSD) is similar in performance, identifying 90% of the high-affinity ecDHFR inhibitors and 77% of the less potent inhibitors; however, this model also hits 26% of the drug-like non-inhibitors. It is only when the simulation time is increased to 4 ns that an improvement in open-loop model performance is observed. The optimal 4-ns model (6 from 7, radii = 2.3 × RMSD) hits a comparable number of inhibitors from both the high-affinity ecDHFR (86%) and general DHFR (63%) inhibitor sets, and there is a marked decrease in the percentage of non-inhibitors being falsely identified (13%).

It is encouraging to see that increased sampling of conformational space, by extending the simulation time from 2 ns to 4 ns, can improve the relatively poor open-loop model by reducing the percentage of false positives identified. Recent work by Boehr *et al.* used NMR relaxation dispersion to study ecDHFR to develop a dynamic energy landscape of catalysis.[9] They found that each intermediate in the catalytic cycle was comprised of a dominate conformation of the protein plus one or two additional conformational states with much lower populations. These higher-energy states resembled the neighboring intermediate in the cycle. This implies that ligand binding occurs by a conformational selection.[34] A small percentage of pre-existing conformations, resembling the bound conformation, coexist in conjunction with the unbound state. A ligand can then bind to the higher-energy conformation, causing an equilibrium shift towards the ligand-bound conformation, which then becomes predominant. The work also suggested that ligand release works in a similar fashion, with the protein adopting a higher-energy conformation resembling the unbound state, before ligand dissociation had occurred.[9] It is possible that extending the dynamic simulations allows more of the "excited states" resembling the closed M20 loop to be sampled, which improves the models performance.

Improvement with increased dynamic sampling was also seen with HIV-1 protease where it was shown that extending the simulation length resulted in identifying more true inhibitors and fewer false positives.[17] This also adds substantiation to our earlier finding with human, *P. carinii*, and *C. albicans* DHFR where models based on crystal structures with the greatest structural variation identified the most high-affinity inhibitors and the fewest false positives compared to those with reduced flexibility.[19] The optimal pharmacophore models from the closed-loop simulations are similar in performance to those generated from screening the *P. carinii* and human DHFR models from our previous work,[19] identifying over 86% of true inhibitors and very few false positives. However, the models from the open-loop dynamics are less successful, with the models identifying more false positives. This is not unexpected as the M20 open-loop conformation does not represent a productive bound complex form.

X-ray pharmacophore model comparison. The pharmacophore model resulting from the X-ray structures (Fig. 6a) shares five elements with the MD-based models: the aromatic element at the bottom of the pocket, hydrogen-bond donors to Asp27 Oδ1 and Oδ2, the aromatic element midway in the active site, and the hydrogen-bond acceptor in the mouth of the binding pocket. In addition, there are two extra hydrogen-bond elements and one extra aromatic element. One hydrogen-bond element is a result of a second cluster of methanol probes interaction with Asp27 Oδ1, and the other reflects an interaction with the backbone carbonyl of Ile94. The additional aromatic element corresponds to the *p*-aminobenzoate linker region of the substrate interacting with Phe31.

The performance of pharmacophore models generated from X-ray crystallographic structures is given in Fig. 6b. The model does identify general DHFR inhibitors over non-inhibitors, but the species-specificity is lost, with a higher percentage of general DHFR inhibitors being identified over high-affinity, species-specific inhibitors. The overall performance is similar to models we previously derived for *C. albicans* DHFR,[19] which also showed little conformational flexibility over the collection of crystal structures. In both cases, although the rate of false positive identification is low, less than 50% of the high-affinity true inhibitors are identified. Again, we see that limited flexibility in the active site leads to modest performance. Even though conformational variability is seen in the M20 loop, it only contributes a small part of the binding site; all other regions of the binding pocket are nearly identical.

The optimal X-ray model (5 from 8, radii = 4.3 × RMSD) identified 42% of the high-affinity ecDHFR inhibitors and 65% of the less potent DHFR inhibitors; only 3% of the false positives were hit. Although this model is able to identify true positives over decoy molecules, it lacks the species-specificity seen in the MD pharmacophore models. The low level of conformational variance in the active site may make the pharmacophore model resemble a static pharmacophore model. The number of extraneous sites and the small radii (low RMSDs) of the elements is characteristic of models produced from a single rigid structure. This again highlights the importance of including sufficient protein flexibility in order to produce selective pharmacophore models for structure-based drug discovery.

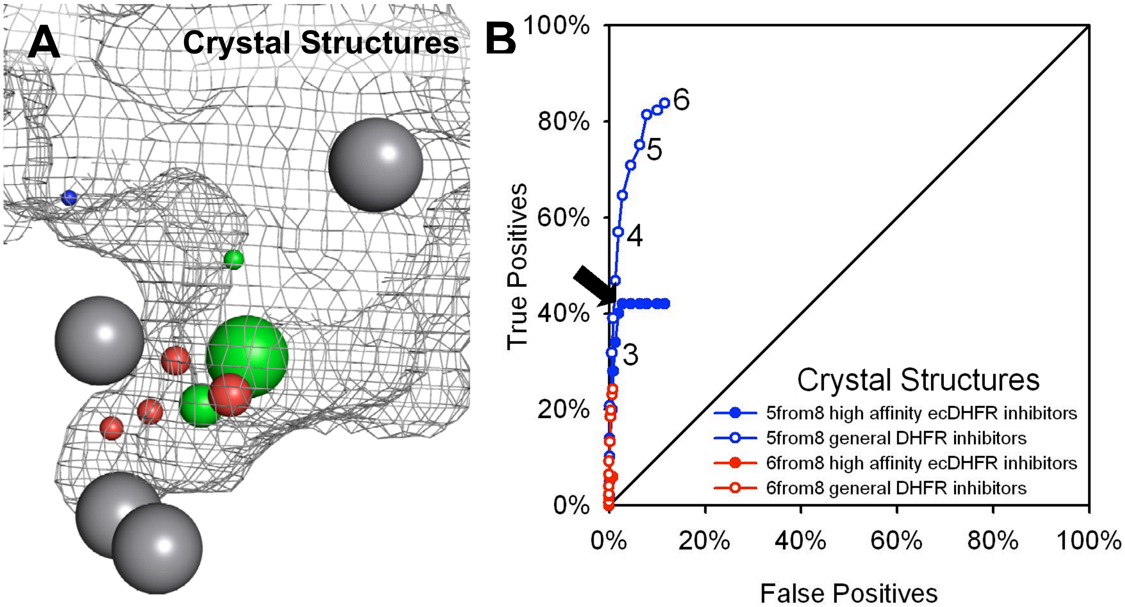


Figure 6. (a) **Pharmacophore model resulting from an ensemble of ecDHFR crystal structures.** Coloring is the same as in Fig. 4. Elements are shown with radii = 2×RMSD. (b) Associated ROC curves for the X-ray pharmacophore model. Series with filled data points are results from the screening of a high-affinity ecDHFR database; those with open data points are results from the screening of general DHFR inhibitors. Shown in red are results from models requiring 6 hits from an 8-site pharmacophore model, shown in blue are results from models requiring 5 hits from an 8-site pharmacophore model. Points along each series represent an increase in pharmacophore element radii from 1×RMSD to 6×RMSD in increments of 1/3.

To our knowledge, the models presented here are the first receptor-based pharmacophore models of *E. coli* DHFR. Joseph-McCarthy and Alvarez have made one based on a single crystal structure of *L. casei* DHFR with MCSSS2PTS.[35,36] MCSS2PTS uses Multiple Copy Simultaneous Search (MCSS),[37] a technique similar to MUSIC, to generate pharmacophore points. For comparison, we have created a model based on a single, static structure taken at equilibration of the closed-loop simulation (Fig. 7). In direct analogy to our MPS models, the radius of an element is defined as the RMSD of the probe cluster that it represents, rather than the RMSD of the consensus cluster.

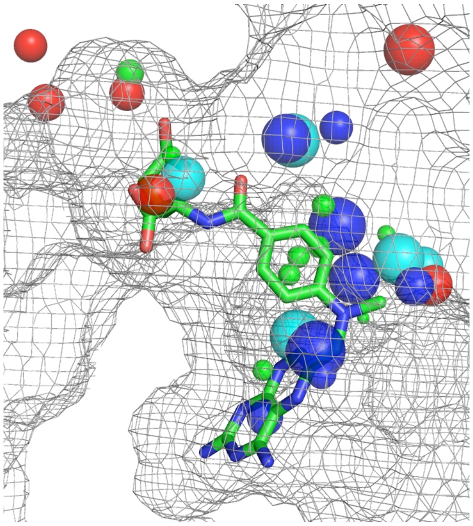


Figure 7. Static pharmacophore model for the equilibrated 1RX1 (M20 closed-loop) structure. Coloring is the same as in Fig. 6. The radius of an element is defined as the RMSD of the probe cluster that it represents, analogous to the RMSD of an MPS consensus cluster. Elements are shown with radii of 1×RMSD. The bound conformation of methotrexate is superimposed on the model to orient the reader.

Although the two techniques are not exactly the same, we find that our model looks very similar to the MCSS2PTS model. In both our static model and the one proposed by Joseph-McCarthy and Alvarez, there are an overwhelming number of pharmacophore sites, only some of which overlap with the features of known ligands. Joseph-McCarthy and Alvarez highlighted the elements that overlap with known binding features of methotrexate. Our use of the MPS naturally highlights these same elements over the many extraneous sites. This is a clear demonstration of how the addition of protein flexibility helps us to identify the key features of the binding site.

Conclusion

From MD simulations of *E. coli* DHFR·NADPH, we have developed receptor-based pharmacophore models that take advantage of protein flexibility. Models resulting from ecDHFR with the closed conformation of the M20 loop were highly selective and identified relevant *E. coli* specific, high-affinity inhibitors over other general DHFR inhibitors. A marked improvement was seen in the open-loop dynamics with an increase in simulation length, indicating that including more flexibility can enhance the pharmacophore models. It is important to note that this inclusion of greater protein flexibility does not result in a loss of specificity, as the optimal models preferentially identify potent ecDHFR inhibitors over other general DHFR inhibitors. These results are particularly encouraging given that the DHFR·NADPH structure does not bias the folate-binding site towards a conformation associated with any particular bound ligand. We have also developed pharmacophore models based on ligand-bound crystal structures. While these models are very selective for DHFR inhibitors, they lack the preference for species-specific, high-affinity inhibitors seen in our models from an ensemble of MD snapshots. Again, this demonstrates the importance of flexibility. The current interest in ecDHFR dynamics in relation to catalysis can only emphasize the importance of including protein flexibility in the quest for identifying novel inhibitors.

Acknowledgements. I thank Dr. Anna L. Bowman for her help throughout this project. This work has been supported by the National Institutes of Health (Grant GM65372) and the Beckham Young Investigator Program. I thank the University of Michigan’s Molecular Biophysics Training Program for support (GM008270). For their generous donation of software I thank OpenEye for OMEGA and CCG for MOE. I also thank Allen Bailey for maintaining the computers used in this work.

Supporting information available. The *E. coli* inhibitor data set used in this work and the coordinates and RMSD of the pharmacophore elements for all models. This material is available in Appendix 1.

References

(1) Voet, D.; Voet, J. In *Biochemistry*; 2nd ed. ed.; John Wiley & Sons, Inc.: New York, NY, 1995, p 762.

(2) Schweitzer, B. I.; Dicker, A. P.; Bertino, J. R. *FASEB J* **1990**, *4*, 2441-2452.

(3) Parker, C. N. *J Biomol Screen* **2005**, *10*, 647-648.

(4) Lang, P. T.; Kuntz, I. D.; Maggiora, G. M.; Bajorath, J. *J Biomol Screen* **2005**, *10*, 649-652.

(5) Radkiewicz, J. L.; Brooks III, C. L. *J Am Chem Soc* **2000**, *122*, 225-231.

(6) Agarwal, P. K.; Billeter, S. R.; Rajagopalan, P. T. R.; Benkovic, S. J.; Hammes-Schiffer, S. *Proc Natl Acad Sci USA* **2002**, *99*, 2794-2799.

(7) Agarwal, P. K.; Billeter, S. R.; Hammes-Schiffer, S. *J Phys Chem B* **2002**, *106*, 3283-3293.

(8) Feeney, J. *Angew Chem, Int Ed* **2000**, *39*, 290-312.

(9) Boehr, D. D.; McElheny, D.; Dyson, H. J.; Wright, P. E. *Science* **2006**, *313*, 1638-1642.

(10) Osborne, M. J.; Schnell, J.; Benkovic, S. J.; Dyson, H. J.; Wright, P. E. *Biochemistry* **2001**, *40*, 9846-9859.

(11) Sawaya, M. R.; Kraut, J. *Biochemistry* **1997**, *36*, 586-603.

(12) Hammes-Schiffer, S.; Benkovic, S. J. *Annu Rev Biochem* **2006**, *75*, 519-541.

(13) Miller, G. P.; Benkovic, S. J. *Chemistry & Biology* **1998**, *5*, R105-R113.

(14) Schnell, J. R.; Dyson, H. J.; Wright, P. E. *Annu Rev Biophys Biomol Struct* **2004**, *33*, 119-140.

(15) Carlson, H. A.; Masukawa, K. M.; McCammon, J. A. *J Phys Chem A* **1999**, *103*, 10213-10219.

(16) Carlson, H. A.; Masukawa, K. M.; Rubins, K.; Bushman, F. D.; Jorgensen, W. L.; Lins, R. D.; Briggs, J. M.; McCammon, J. A. *J Med Chem* **2000**, *43*, 2100-2114.

(17) Meagher, K. L.; Carlson, H. A. *J Am Chem Soc* **2004**, *126*, 13276-13281.

(18) Meagher, K. L.; Lerner, M. G.; Carlson, H. A. *J Med Chem* **2006**, *49*, 3478-3484.

(19) Bowman, A. L.; Lerner, M. G.; Carlson, H. A. *J Am Chem Soc* **2007**, *129*, 3634-3640.

(20) Damm, K. L.; Carlson, H. A. *J Am Chem Soc* **2007**, ASAP DOI: 10.1021/ja0709728.

(21) DeLano, W. L. **2002**, DeLano Scientific, Palo Alto, CA, The PyMOL Molecular Graphics System v0.99.

(22) Pearlman, D. A.; Case, D. A.; Caldwell, J. W.; Ross, W. S.; Cheatham III, T. E.; Debolt, S.; Ferguson, D. M.; Seibel, G. L.; Kollman, P. A. *Comput Phys Commun* **1995**, *91*, 1-41.

(23) Holmberg, N.; Ryde, U.; Bulow, L. *Protein Eng* **1999**, *12*, 851-856.

(24) Jorgensen, W. L.; Chandrasekhar, J.; Madura, J. D.; Impey, R. W.; Klein, M. L. *J Chem Phys* **1983**, *79*, 926-935.

(25) Case, D. A.; Pearlman, D. A.; Caldwell, J. W.; Cheatham, I., T E ; Ross, W. S.; Simmerling, C. L.; Darden, T. A.; Merz, K. M.; Stanton, R. V.; Cheng, A. L.; Vincent, J. J.; Crowley, M.; Tsui, V.; Radmer, R. J.; Duan, Y.; Pitera, J.; Massova, I.; Seibel, G. L.; Singh, U. C.; Weiner, P. K.; Kollman, P. A. **1996**, University of California, San Francisco, San Francisco, CA, AMBER 6

(26) Summerfield, R. L.; Daigle, D. M.; Mayer, S.; Mallik, D.; Hughes, D. W.; Jackson, S. G.; Sulek, M.; Organ, M. G.; Brown, E. D.; Junop, M. S. *J Med Chem* **2006**, *49*, 6977-6986.

(27) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. *Nucleic Acids Res* **2000**, *28*, 235-242. http://www.rcsb.org/pdb/.

(28) Jorgensen, W. L. **2000**, Yale University, New Haven, CT, BOSS Version 4.2.

(29) Jorgensen, W. L.; Maxwell, D. S.; Tirado-Rives, J. *J Am Chem Soc* **1996**, *118*, 11225-11236.

(30) Damm, K. L.; Carlson, H. A. *Biophys J* **2006**, *90*, 4558-4573.

(31) **2005**, Chemical Computing Group, Montreal, Canada, MOE v2005.06.

(32) **2005**, MDL Information Systems Inc., San Leandro, CA., Comprehensive Medicinal Chemistry

(33) **2004**, OpenEye Scientific Software, Inc., Santa Fe, NM, OMEGA Version 1.8.b1.

(34) Tsai, C. J.; Kumar, S.; Ma, B. Y.; Nussinov, R. *Protein Sci* **1999**, *8*, 1181-1190.

(35) Joseph-McCarthy, D.; Alvarez, J. C. *Proteins* **2003**, *51*, 189-202.

(36) Joseph-McCarthy, D.; Thomas, B. E.; Belmarsh, M.; Moustakas, D.; Alvarez, J. C. *Proteins* **2003**, *51*, 172-188.

(37) Miranker, A.; Karplus, M. *Proteins* **1991**, *11*, 29-34.

Chapter 3

Correlated and conformational dynamics of the DHFR•NADPH complex

Introduction and background

Dihydrofolate reductase (DHFR; EC 1.5.1.3) catalyzes the NADPH-dependent reduction of dihydrofolate (DHF) to tetrahydrofolate (THF). DHFR is the only source of tetrahydrofolate, an important precursor in the biosynthesis of purines, thymidylate and several amino acids. Thus, it has been a long-standing anti-cancer target.[1] The potential to target species-specific DHFR has also made it an important antibiotic and anti-microbial target.[2] The full catalytic cycle is summarized in Fig. 8. The structure, dynamics, and function of DHFR have been studied extensively, as summarized in a recent review,[2] and flexibility has been shown to be of paramount importance throughout the catalytic cycle. Correlated dynamics have been shown to contribute to catalysis and to change in distinct patterns throughout the catalytic cycle.[3] Most recently, NMR experiments have shown that each catalytic intermediate samples so-called "excited states," which are conformations similar to the adjacent states in the catalytic cycle.[4]

DHFR is composed of an eight-stranded β-sheet (βA - βH) and four α-helices (αB, αC, αE, and αF). It has two rigid-body subdomains, the adenosine-binding domain and the loop subdomain. The conformation of the M20 loop in the loop subdomain is particularly important to the catalytic cycle, and it is known to regulate ligand binding, selectivity, and turnover.[2,5-12] Three conformations of the M20 loop are known: open, closed, and occluded. The Michaelis complex DHFR•NADPH•DHF is found with the M20 loop closed. When the product is bound (DHFR•NADP+•THF, DHFR•THF, or DHFR•NADPH•THF), the M20 loop is found in the occluded position, forcing the nicotinamide moiety of the cofactor out of the pocket. Meanwhile, DHFR•NADPH is found in both the open and closed conformations. Fluctuations of the M20 loop are on the same time scale as substrate/cofactor binding and product release, and they are thought to guide the protein through the catalytic cycle.[2] The M20 loop has also been shown computationally to exhibit a "tightly closed" variation of the closed conformation that compresses and stabilizes the active site, contributing directly to catalysis.[7]

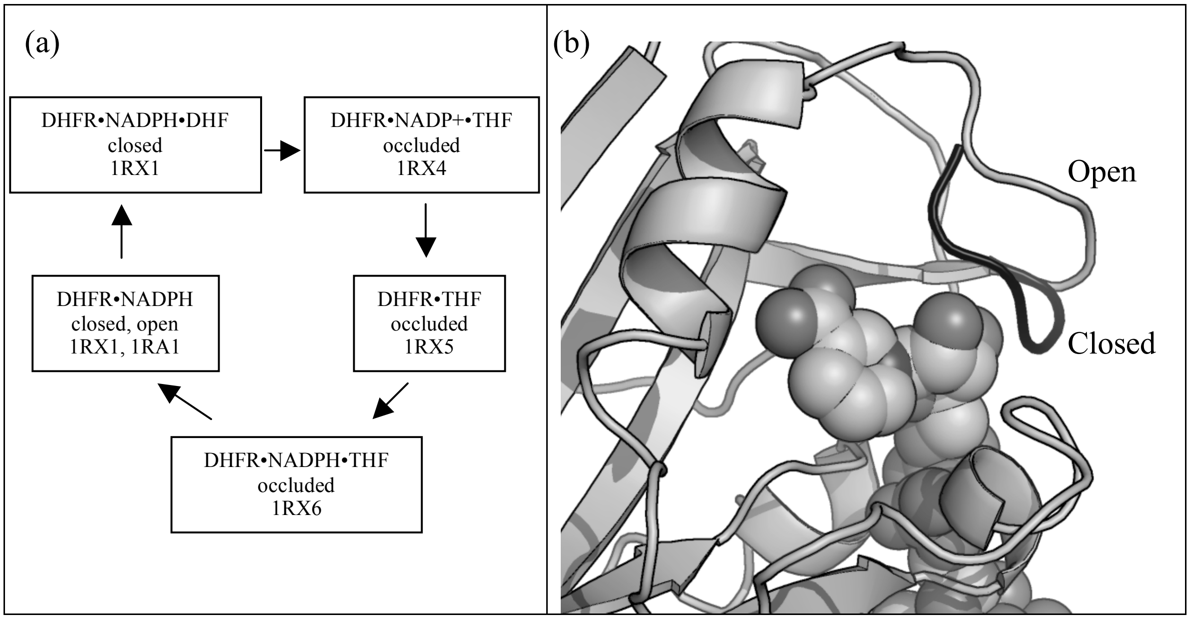


Figure 8. The catalytic cycle and the M20 loop of DHFR. (a) DHFR's catalytic cycle, indicating M20 loop conformations and representative crystal structures.[4,6,13] (b) The open (grey) and closed (black) conformations of the M20 loop seen in the DHFR•NADPH structures. Non-hydrogen atoms of NADPH are shown in spheres.

NMR experiments have shown that ligand and cofactor binding changes flexibility and dynamic motions throughout the protein, both in the active site and distally.[4,10,14] The motional coupling of DHFR has been further studied through both experiment and theory. Several mutagenesis studies have indicated residues near and far from the active site that have a significant effect on catalysis, with Ala9, Asp27, Leu28, Phe31, Arg44, His45, Thr46, Leu54, Tyr100, Thr113, Gly121, and Asp122 affecting the first three steps of the catalytic cycle by a factor of 6 or more.[3,9,15-26] The fact that distal mutations have nonadditive effects supports the notion that long-range coupling exists between the residues and enhances catalysis. In particular, hydride transfer is affected by motions of the FG and M20 loops on the picosecond-nanosecond time scale.[27] Mutations of Gly121 have been shown to alter the ensemble of substates sampled by the protein, thus increasing the hydride transfer barrier.[11]

Radkiewicz and Brooks used classical molecular dynamics (MD) to study these coupled motions via correlated dynamics analysis of DHFR•DHF•NADPH (Michaelis complex), DHFR•THF•NADP+ (product ternary complex), and DHFR•THF•NADPH (product release complex).[3] While some level of correlated dynamics was found in all states, the correlations were only significant in the reactant state, DHFR•DHF•NADPH. The regions of strong anti-correlation were particularly enlarged in this state, and almost nonexistent in the other states. They examined 11 of the 12 mutations listed above (Asp27 was ignored, as it is directly involved in the chemical reaction). Four of those mutations are in the active site and are not involved strongly correlated dynamics. The remaining seven are all found in regions of strong anti-correlated motion. Catalytically important mutations have been shown to alter the pattern of correlated dynamics.[28] Wong *et al.* have also shown that residues in several distal regions (including the M20, FG, and GH loops and the adenosine-binding domain) affect the network of coupled motions.[29] Recently, hybrid quantum/classical molecular dynamics (QM/MM) studies of both *E. coli* and *Bacillus subtilis* DHFR have been compared to the classical MD studies of *E. coli* DHFR, and these studies found similar correlated dynamics in the reactant, transition, and product states.[30,31]

In this paper, we use classical MD to study the conformational changes and correlated dynamics of the *E. coli* DHFR•NADPH complex. We perform these simulations for two DHFR•NADPH complexes, one with an open M20 loop and one with a closed-loop starting structure. To our knowledge, these are the first MD simulations of DHFR•NADPH in the literature. We show that conformational states pre-exist that mimic the binding of DHF, in good agreement with Boehr *et al.*[4] More importantly, we find that the dynamics seen in other theoretical studies also pre-exist transiently in our simulations! The theory of pre-existing conformational states is gaining wide acceptance, and our findings of pre-existing dynamics further develops these concepts. Detailed structural analysis is presented to support these findings. This analysis has also revealed a potential allosteric site in DHFR.

Methods

Molecular dynamics

These studies are an extension of a previous examination of the suitability of DHFR•NADPH ("apo") structures for structure-based drug design.[32] Those studies used snapshots from 4 ns of simulations; here, we have extended the simulations to 10 ns to better examine the dynamic behavior. The closed- and open-loop crystal structures of the

DHFR•NADPH complex (accession codes 1RX1[6] and 1RA1[6] respectively) were obtained from the PDB.[33] In the open-loop complex, the NADPH's nicotinamide is mobile, moving in and out of the active site. Thus, it is unresolved in the 1RA1 crystal structure. When the M20 loop is excluded, the root-mean-square deviation (RMSD) between 1RA1 and 1RX1 is 0.461 Å. Given this high degree of structural similarity, the rest of the cofactor was modeled into the 1RA1 structure by direct comparison with the 1RX1 coordinates. The electron density indicates that the nicotinamide is found in the pocket 75% of the time in the open-loop structure, and 100% of the time when the substrate binds and the M20 loop closes.[6] This conformation was chosen because it is both more suited to modeling substrate binding.

A custom PyMOL[34] script was used to check side-chain orientations and protonation states. The LEaP routine from the AMBER[35] suite of programs was used to place hydrogen atoms, and conjugate gradient energy minimization (convergence criterion: RMS gradient < 0.0001 kcal/mol Å) was used to optimize their positions. Parameters for the protein were taken from the AMBER94[36] force field, while parameters for the cofactor were obtained from the literature.[37] The DHFR•NADPH complex was solvated with explicit TIP3P[38] waters in a truncated octahedral box extending 15 Å from the protein surface. MOE[39] was used to generate an electrostatic potential surface 10 Å from the protein surface, and a single sodium ion was placed at the minimum on that surface. This procedure was repeated until a total of 15 counter-ions had been added, thus giving the system a net neutral charge. The system was then subjected to 300 steps of conjugate gradient minimization to remove bad initial contacts. The protein heavy atoms were held fixed while the water molecules were heated at constant volume from 10 K to

310 K over 50 ps. This was followed by 300 ps of constant-pressure equilibration. The heavy-atom restraints were then removed and the entire system was heated from 10 K to 310 K over 30 ps and subsequently equilibrated for 50 ps of equilibration at 310 K, all at constant volume. The final phase of equilibration was 1.5 ns at constant pressure. Previous studies have found that DHFR requires longer equilibration, ranging from 0.9 ns to 2.5 ns.[3] We also found that a long equilibration was necessary; both the open- and closed-loop simulation were equilibrated and stable after 1.5 ns.[32] For both conformations of DHFR, a total of 10 ns of NPT sampling was collected for analysis.

Correlated dynamics

The ptraj module of AMBER 8[40] was used to calculate correlation matrices for various parts of our trajectories. As our methods are most similar to the original work by Radkiewicz and Brooks,[3] we have emulated their color scheme in which strongly anti-correlated residues are dark blue, uncorrelated residues are light blue, correlated residues are yellow and strongly correlated residues are red. We have examined the correlation by α-carbons, as is the standard in the literature,[3] but we also present a more detailed analysis of all main-chain heavy atoms and the cofactor. The plots presented in this paper were created by examining windows of size 1 ns at intervals of 100 ps throughout the trajectory, labeled in the style "1.2-2.2 ns", "1.3-2.3 ns", "1.4-2.4 ns", etc.

Hydrogen bonds and bridging water

The ptraj module of AMBER 8[40] was used to calculate intraprotein and protein-cofactor hydrogen bonds throughout the trajectories. Default parameters for distance and angle cutoffs were used (3.0 Å and 120° respectively).

Anisotropic thermal diffusion (ATD)

The original techniques for studying ATD were developed by Ota and Agard.[41] We have implemented a similar protocol using AMBER 8.[40] The LES module was used to selectively heat individual residues. As per the work of Ota and Agard, a super-cool equilibration of the entire system was performed; harmonic restraints of 5 kcal/mol-Å^2 were applied to backbone and exposed atoms, and a distance-dependent dielectric (ε=4r) was used. With AMBER, the selected residue is coupled to a high-temperature heat bath, and the rest of the protein is coupled to a low-temperature heat bath. This coupling of the rest of the protein redistributes energy so, rather than comparing absolute values of the calculated B-factors, we compared them relative to the B-factor of the excited residue. Excited residues were placed into groups having B-factors $\geq$ 1/3 that of the excited residue, $\geq$ 2/3 that of the excited residue, etc.

Pocket analysis

Three programs were used in the analysis of the new pocket: CAVER,[42] the fuzzy-oil-drop (FOD) method,[43,44] and MOE[39] Site Finder.[45] Given a starting point in the interior of a protein, CAVER calculates open pathways to the outside solvent. It is implemented both as a stand-alone program and as a plug-in for PyMOL[34]. CAVER uses a grid-based method. Pathways are reported as a series of grid points, along with the maximally sized sphere that can be inscribed at each point. The distributed PyMOL plug-in works only with static structures, and we modified it to report information throughout a trajectory. We calculated a maximum of 10 pathways through each snapshot. Default values were used for all other parameters.

The FOD method[43,44] uses hydrophobic deficiency – the difference between empirical hydrophobicity and that predicted by a 3D Gaussian function, also known as a

FOD model – to predict functional sites in a protein. The method has been tested with several classes of enzymes. The general principle is that functional sites are more hydrophobic than an idealized FOD model would predict. The method is available for use via a web-based interface (http://loschmidt.chemi.muni.cz/caver/index.php). Default values were used for all parameters, including the use of the FOD hydrophobicity scale.

MOE Site Finder[45] is an automated tool based on alpha spheres that uses a combination of geometry and hydrophobicity to detect and rank potential binding sites on a protein surface. MOE Site Finder is included with MOE,[39] and default values were used for all parameters.

Results and discussion

Preorganization of the binding site

During the MD simulations of DHFR in the absence of folate, pre-organization of the binding site is observed. Arg 52, Arg57 and the M20 loop adopt conformations that clearly show conformations related to substrate binding and catalysis, Fig. 9.

To identify conformations similar to other steps in the catalytic cycle, all-atom RMSD analysis was used to compare conformations from the trajectories to crystal structures representing states in the catalytic cycle. Arg52 showed clear conformational changes during both the open- and closed-loop simulations. Its side chain switches between two stable conformations, one orientation pointing in toward the folate-binding site. These conformations can be clearly distinguished in various crystal structures that show the inward conformation makes two important hydrogen bonds with folate. Additionally, Arg57 contributes significantly to the overall variation within the binding-site residues. Arg57 makes key hydrogen bonds with folate, and it is pre-organized for folate binding in the crystal structures. In the MD, Arg57 sampled several states, coming in and out of the pre-organized state, clearly showing that the "bound conformation" is one of several available in the unbound state. Wright and coworkers have used NMR experiments to identify marker residues that, independent of M20 loop conformation, undergo significant and reliable shifts upon folate binding.[4,14] Arg52 and Arg57 are among those marker residues.

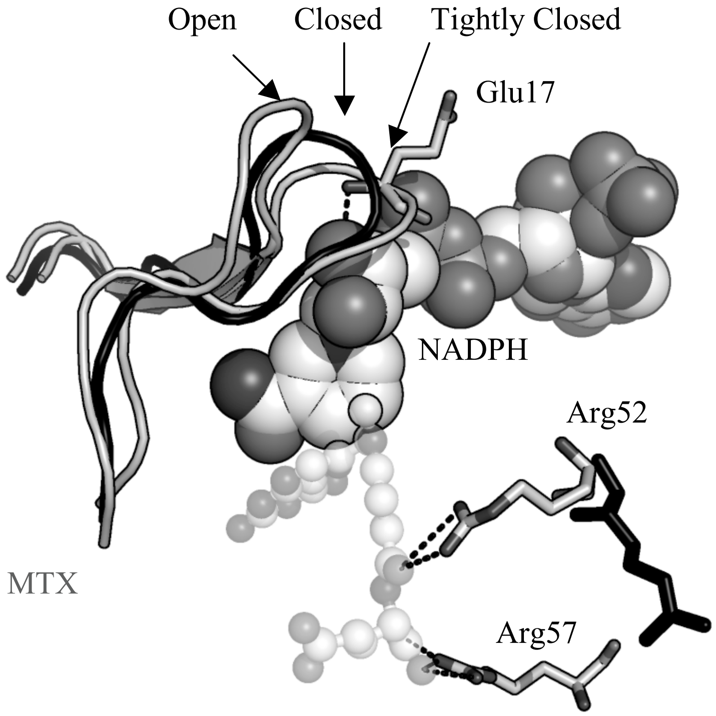


Figure 9. Preorganization of the binding site observed during the two MD simulations. The M20 loop conformations sampled three conformations: open, closed, and "tightly closed". The backbone of Glu17 makes a key hydrogen bond to NADPH in the tightly-closed conformation. The position of methotrexate is shown in transparent ball-and-stick to highlight ligand interactions in the bound state (taken from the 1RA3 crystal structure – we emphasize that no folates were present during the simulations). Arg52 samples conformations suitable for hydrogen bonding with bound ligands, but its starting position from the 1RX1 crystal structure (black) cannot make these interactions. Arg57 starts in the pre-organized orientation and samples in and out of this conformation.

Analysis of the trajectories shows clear differences in the loop conformations. This is in agreement with previous studies that have also reported transitions between loop conformations during MD simulations.[3,28] During the closed-loop simulation, a new conformation is observed in which the M20 loop packs even more tightly against the NADPH. In addition to generally tighter packing, this new conformation (Fig. 9) is also characterized by the formation of a hydrogen bond between NADPH and the Glu17 backbone. This tighter packing may be relevant to catalysis; recently, Khavrutskii *et al.* have reported a "tightly closed" conformation that has been shown to contribute to catalysis by enhancing the interaction of the substrate and cofactor.[7] While they do not provide explicit details as to the exact conformation of this tightly closed state, we believe that the tightly closed conformation seen in our simulations may be similar. During the open-loop simulation, we see the M20 loop adopt a "more open" conformation. In both simulations, the other loops exhibit significantly less conformational flexibility, and they are known to have significantly less differentiation between various stages of the catalytic cycle.

Correlated dynamics and the network of coupled residues

While pre-organized conformational states were recently indicated by NMR,[10,27,46-48] our simulations have revealed for the first time that dynamics unique to the reactive state also pre-exist the binding of folates. Previous calculations by Brooks and coworkers[3,28] indicated that these dynamics are unique to the reactant state, but it is possible that the dynamics were not observed because their simulations were analyzed over 10 ns. FG and M20 loop motions on the picosecond-nanosecond time scale are known to affect hydride transfer,[10,27,46] and QM/MM studies have shown that a network of coupled motions on the femtosecond-picosecond time scale affect hydride transfer.[49] Therefore, it is important to analyze the simulations on time scales that will preserve information about these motions.

The complex in general. When correlations are calculated over the entire 10 ns of our MD, no significant regions of strong anti-correlated motion are observed. The results look similar to Radkiewicz and Brooks's[3] product state (DHFR•NADP+•THF) simulations. However, when analyzed in 1-ns segments, our simulations clearly show strong anti-correlated dynamics from the reactant state (DHFR•DHF•NADPH). The major regions of strong anti-correlated dynamics include the M20 loop (anti-correlated with αC, the CD loop, αE, and αF), the FG loop (anti-correlated with αC, the CD loop αE, and αF), and αF (anti-correlated with the M20 loop, the CD loop, αE, and the FG loop). It is intriguing that these dynamics are present *transiently* throughout our simulations, as shown in Fig. 10. In the closed-loop simulation, the motions appear to become activated quite suddenly around 0.5 ns and then fluctuate throughout the rest of the simulation. We will refer to this phase of the simulation as "activated dynamics." In the open-loop simulation, the strength of the correlated motions fluctuates throughout the entire simulation. The correlations clearly show that the M20 and FG loops consist of separate sub-regions. The beginnings and ends of these loops are defined by crystallographic considerations. However, different subsets of these loops are in contact with different parts of DHFR's secondary structure and have different degrees of exposure to solvent; this results in the sub-regions having different dynamic behavior.

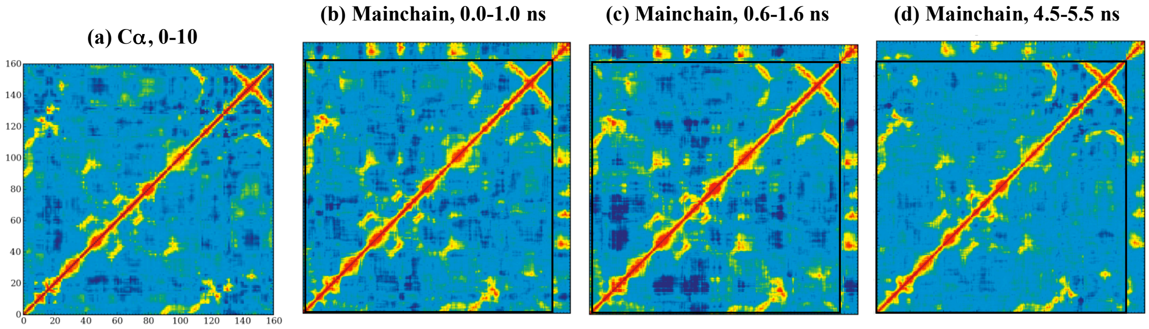


Figure 10. Pre-existing dynamics. (a) Cα correlated dynamics for the entire closed-loop simulation does not reveal the key motions related to catalysis. (b, c, d) When examined in 1-ns windows, correlated dynamics of main-chain, heavy atoms show the transient nature of the strong correlated motions of DHFR (within black box) and NADPH. The color scheme is detailed in Fig. 11.

The dynamics resemble those previously reported for wild-type DHFR•DHF•NADPH simulations, rather than those from the DHFR•THF•NADP+ and DHFR•THF•NADPH[3] or the less catalytically active DHFR mutants.[28] NMR experiments on CypA have detected a similar effect, where enzyme motions from the catalytic step are found in the unbound state; furthermore, they are found with frequencies corresponding to turnover rates.[50] The window from 0.6-1.6 ns shown in Fig. 11 – representative of the activated dynamics in our simulations – includes all strongly anti-correlated regions seen in Radkiewicz and Brooks's DHFR•DHF•NADPH simulation,[3] with the exception that we see a diminished anti-correlation in the M20-αE region. We do see strong anti-correlation in this region later in the simulation (e.g., the 4.2-5.2 ns window). We see all of Radkiewicz and Brooks's regions of strong positive correlation, but we observe a stronger positive correlation between aF and Ile5 and Ala6.

In fact, the entire section becomes a correlated block on several occasions (e.g., 6.5-7.5 ns).

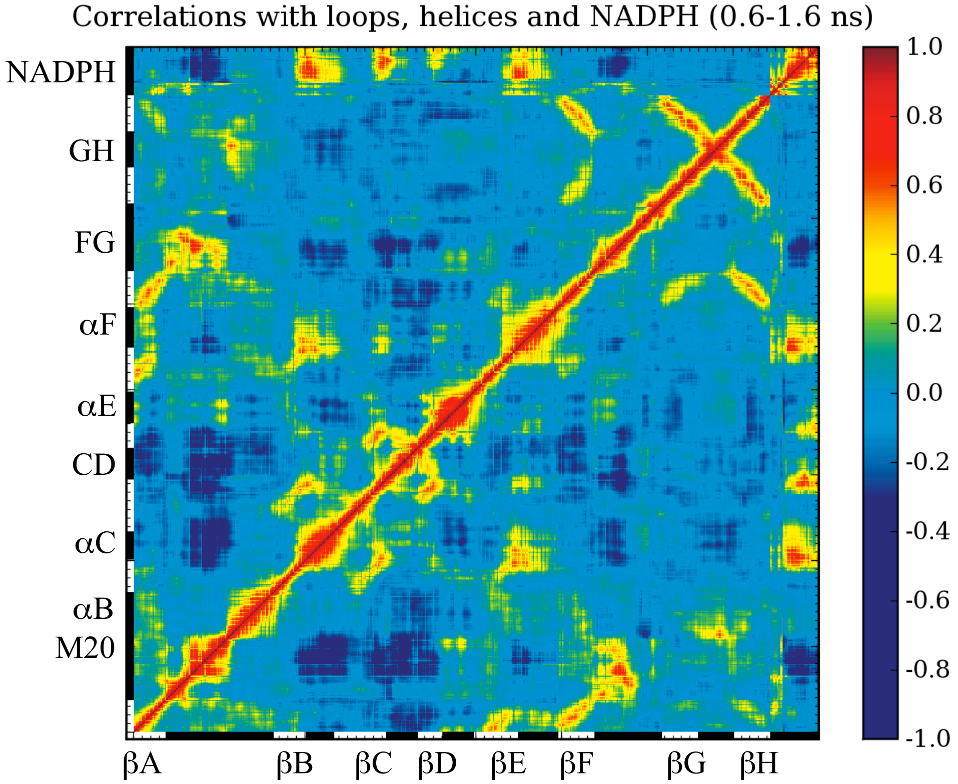


Figure 11. Representative period of strong correlations between loops, helices, and the cofactor. The scale ranges from -1 (dark blue, fully anti-correlated) to +1 (dark red, fully correlated). The regions of strong anti-correlated motions involve primarily the loops. The correlations clearly show that the M20 and FG loops have separate subregions that have different behavior. The cofactor exhibits regions of strong positively- and negatively-correlated motion. The motion of the cofactor is shown to be anti-correlated only with regions of the key M20 and FG loops.

Our activated dynamics are also similar to those observed by Hammes-Schiffer and coworkers[30,31] when following the reaction coordinate from reactant to transition state to product for both *E. coli* and *Bacillus subtilis* DHFR. They note that, in general, dynamics are stronger in the reactant state. The similarity is qualitative, rather than quantitative, because the energy surfaces sampled in our MD is different than that of the reactant state in Hamer-Schiffer's QM/MM simulations.[30,31] They note that the correlation between residues 117-120 of the FG loop and 87-90 of the hinge region is positive in the reactant state and negative in the transition state. They also note that residues 67 and 55 are positively correlated in the reactant state and negatively correlated in the product state. They note further that the presence of anti-correlations between the GH and M20 loops may be used to distinguish the reactant and transition states. Finally, Hammes-Schiffer and coworkers find a significant island of motion between the FG and GH loop in the reactant state (unseen in studies from the Brooks group[3]). In our simulations, the region involving residues 117-120 and 87-90 is intermittently anti-correlated. We find that residues 67 and 55 are intermittently both positively and negatively correlated in our simulations. We see a transient region of strongly anti-correlated dynamics between the GH and M20 loops, similar to the reactant state of Hammes-Schiffer and coworkers. Finally, we see a significant island of anti-correlated motion between the FH and GH loop, persistent enough in our simulations that it appears in correlation plots derived from the entire trajectory as well as the 1 ns windows. Taken together, our simulations of DHFR•NADPH show strong agreement with the reactive-state dynamics described by comparable simulations of folate-bound DHFR conducted by Brooks and coworkers; they also show intermittent agreement with various characteristics of the reactant, transition and product states described by Hammes-Schiffer and coworkers using a QM/MM modeling technique.

These correlated dynamics are supported by a network of residues exhibiting coupled motions (see Appendix 1). Several studies have been done to elucidate the exact makeup of this network via NMR, MD, and folding experiments.[29-31,49,51-55] The network is theorized to be stabilized by hydrogen bonds[28] and electrostatic effects.[30] Hammes-Schiffer and coworkers have conducted several QM/MM studies to identify a network of coupled residues and promoting motions along the reaction coordinate.[31,49,51] Two earlier papers from Agarwal and coworkers[49,51] explicitly list a network of coupled residues from *E. coli* DHFR studies. More recently, Watney and Hammes-Schiffer performed a QM/MM study of *E. coli* and *B. subtilis* with substrate and cofactor bound.[31] Due to the large differences in time scale, we cannot make direct comparisons with the work of Hammes-Schiffer and coworkers.[29-31] However, we find a surprisingly strong correspondence with their results when we look at residues that exhibit significant anti-correlated motion, as shown in Fig. 11. One difference stands out: although they identify residues that are anti-correlated with the FG and GH loops, they do not find anti-correlations between the FG and GH loops, while we do. While the work of Pan *et al.*[52] focuses on correlations involved in folding and unfolding, we can still make some interesting comparisons. In particular, they find negative correlation between the folate-binding site and residues 63-68, indicating that folate binding induces destabilization of the loop. They verify this experimentally by comparing B-factors for folate-bound and -unbound. With the onset of activated dynamics in our studies, we find that residues 63-68 are anti-correlated to the M20 loop as well as to a large portion of the protein, including residues in the folate- and NADPH-binding sites. These residues are also positively correlated to NADPH itself.

For comparison purposes, we have compiled a list of all residues identified in the literature as being involved in *E. coli* networks. One plot of our correlated dynamics in

Appendix 1 is highlighted, showing that all of these residues are involved in regions of correlated or anti-correlated motions, with most involved in regions of strong anti-correlation. Mutagenesis of residues within the regions of strong anti-correlation have been shown to greatly effect catalysis.[9,15-26,56]

The cofactor. Because we have analyzed the simulation by all heavy atoms, and not just the Cα, particularly interesting correlations can be elucidated with the cofactor (Fig. 11). We see regions of both strong correlation and anti-correlation with loops and helices, and there are clear distinctions between the individual chemical moieties of NADPH. The nicotinamide is always positively correlated to the M20 loop. During the activated dynamics in the closed-loop simulation, this correlation decreases and the adenine moiety becomes strongly anti-correlated with the M20 loop. It is reasonable that these two moieties of NADPH should show differing dynamic relationships to the M20 loop, given that the occluded conformation only forces the nicotinamide moiety out of the binding site. The CD loop is strongly positively correlated to the adenosine and phosphate linker, as are residues 76-78 that comprise a small loop between βD and αE, located near the adenine. The FG loop (defined crystallographically as residues 116-133) exhibits strong patterns of both positive and negative correlation with the cofactor at different times throughout the simulations. During the activated dynamics, it is anti-correlated with the adenosine and the phosphate linker. Later in the trajectory, residues 120-124 become strongly positively correlated with nicotinamide and associated ribose. Lastly, we note the GH loop is strongly anti-correlated to the phosphate linker during the activated dynamics (this anti-correlation begins just after the snapshot shown in Fig. 11), but uncorrelated at most other times. As anti-correlations with the loops dominate the reactant-state dynamics of Radkiewicz and Brooks,[3] it is worth noting that the cofactor exhibits strong anti-correlation with all of the loops at various points in the simulations. In the closed-loop simulation, strong anti-correlation with both the CD and FG loops precedes the activated dynamics. This trend is less clear in the open-loop simulation, which does not display such a distinct onset of strongly correlated dynamics.

The helices also shows significant correlations with NADPH. αC is always positively correlated with the adenosine and phosphate linker and, to a lesser degree, the ribose and nicotinamide. αF is positively correlated to all of NADPH throughout the entire simulation. We are able to identify at least one breathing motion. The phosphate linker and adenosine ribose are sandwiched between αC and αF, both of which hydrogen bond to the cofactor. As might be expected, αC, αF, the phosphate linker, and the adenosine ribose move together as a unit, exhibiting strong positive correlations with each other. Most of the strong intra-protein, anti-correlated motion involves loops, and αC and αF are the only non-loop regions involved in large intra-protein anti-correlations; both are anti-correlated with the M20 loop, as are the ribose and phosphate linker. Thus, we see a breathing motion with the M20 loop on one side and the αC/αF/phosphate linker/ribose block on the other. Finally, we note that, aside from the exceptions noted above, all of these cofactor-protein correlations are greatly strengthened during the activated dynamics.

A potential allosteric binding site

Examination of folate-bound crystal structures reveals the existence of several important hydrogen bonds between the protein and folate, located at the bottom of the active site. However, this part of the binding site is fairly closed off in the folate-free crystal structures, and there is not enough room for the base of the pterin ring. This may explain why our MD simulations and others have shown that longer equilibration times are necessary for this system. During our examination of the relaxation at the base of the folate-binding site, we discovered an unusual pathway through the protein, extending from the folate site to a second, smaller site. The residues that line the second site show unusual physical characteristics and dynamics that lead us to propose this as a potential allosteric site on ecDHFR. This suggestion is supported by older mutagenesis studies and the position of crystallographic additives in one X-Ray structure.

We used CAVER[42] to characterize the shape of the folate binding site throughout our trajectories. CAVER allows for easy identification and visualization of pathways from the interior of a protein to the surrounding solvent. When CAVER is started at the bottom of the folate-binding site, the predominant pathway passes through the region in which folate binds. A secondary cavity was discovered which opened gradually over the equilibration of the closed-loop simulation. Fig. 12 shows how the primary cavity and the secondary pathway form a tunnel that passes through the protein. While the narrowest portion of the cavity is often too tight for water to pass through, it opens into a pocket on the far side that is large enough to bind small molecules. When we examined snapshots taken every 5 ps throughout our simulations, we found that the cavity persists throughout almost all of the closed-loop simulation, while it appears in only one snapshot from the open-loop simulation. This indicates that fluctuations on a larger timescale may be important in controlling the cavity and associated pocket. Another possibility, suggested by the fact that it appears primarily in the closed-loop simulation, is that it is correlated to folate-organized binding sites.

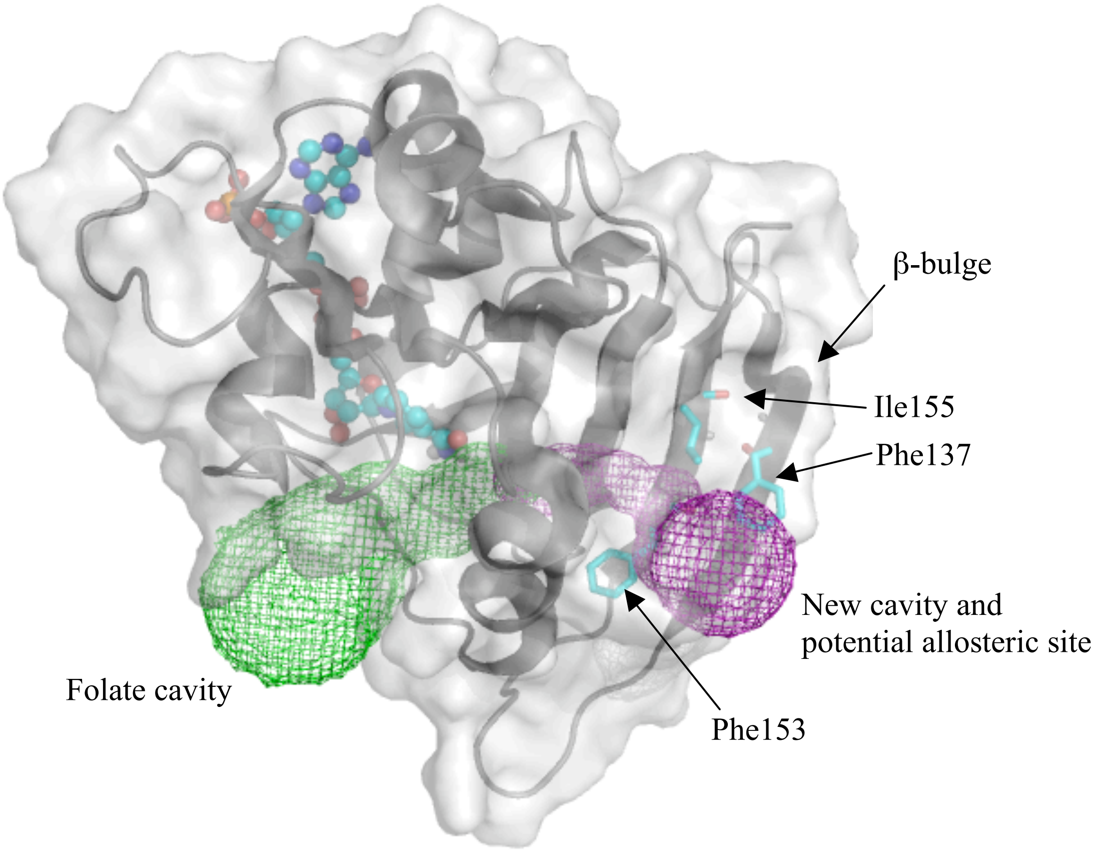


Figure 12. A potential allosteric site. The new site is at the end of the cavity shown in purple mesh. It is flanked by Phe137, Phe153 and Ile155, shown in sticks. The cavity containing the folate-binding site is shown in green mesh, and NADPH is shown in ball-and-stick. The β-bulge is clearly visible above Phe137. The M20 loop and main entrance to the active site is on the rear face of the figure. The green and purple cavities were generated by applying CAVER to the equilibrated closed-loop structure, and it clearly shows the connectivity between the two.

Other codes also identify the new site as a potential binding site. MOE[39] Site Finder[45] uses geometry and hydrophobicity to detect and rank potential binding sites on a protein surface, often breaking a known pocket into several sub-regions. In the examination of our fully equilibrated closed-loop structure, the two highest-ranked sites comprise the folate-binding site. The next two highest-ranked sites comprise the new pocket. The FOD method has previously been used to predict binding sites from PDB structures.[43,44] It uses hydrophobic deficiency to predict functional sites in a protein.

When used to analyze the closed-loop crystal structure 1RX1, FOD revealed that the residues most probable for binding extended from the folate-binding site through the cavity and into the new pocket.

A recent study by Soga *et al.* showed that the amino acid composition of binding sites is significantly different from that of non-binding concavities on the protein surface.[57] They found that Trp, Phe and Tyr are the first, third and fifth most likely amino acids to be found in binding sites, respectively. The surface-exposed residues in the new pocket are Ala26, Ala29, Trp30, Arg33, Tyr111, Phe137, Glu139, His141, Phe153 and Ile155. Thus, the number of aromatic side chains in the pocket indicates that it has a valid composition for a binding site.

It is possible that this new site may be utilized for allosteric control. Experimental evidence indicates that there is clear structural communication between this pocket and the active site. The far pocket contains a β-bulge that was examined via mutagenesis experiments in the early 1990s, but has been largely ignored in more recent work. Double-mutant experiments have shown that Phe137, Phe153, and Ile155 communicate with the active-site residue Asp27 to mediate catalysis and ligand affinity.[20,58] Moreover, crystallographic experiments comparing the wild-type structure to the D27S/F137S mutant have shown that it is likely that there is structural communication between these residues, involving the αB helix and the side chain of Trp30.[59] In these experiments, the Trp30 side chain rotates approximately 180° degrees from the conformation seen in the wild-type crystal structures in order to facilitate this communication. While we see some movement of this side chain, we do not see the 180° rotation. It is possible that this conformation exists only in the mutant structure, or the side chain flips on a timescale longer than that of our simulation. When we consider the entire pathway from the active site to the far pocket, we find an additional residue (Thr113) for which mutation has a very strong affect on catalysis, and several other residues (Trp30, Tyr111, Leu112, His114) that are adjacent to residues for which mutation has a very strong affect on catalysis.

Further experimental evidence of this site's ability to bind small molecules is provided in the crystallographic literature. A crystal structure from *M. tuberculosis* (1DF7[60]) shows DHFR bound to NADPH, MTX, and four glycerol molecules. One of these glycerols is found next to MTX in the active site, and the authors propose its location as a way to extend MTX and develop improved drug leads. A second glycerol, undiscussed by the authors, is found in our proposed allosteric site, and its binding is stabilized by three hydrogen bonds. The fact that it is found in an MTX-bound structure lends support to the idea that this binding site may be related to folate binding.

To further investigate the possibility of allosteric control, we have examined the flow of energy between the far pocket and the active site. Ota and Agard have developed the ATD method to investigate allostery.[41] Select residues are heated, and the diffusion of thermal energy is observed. A directed (anisotropic) flow of energy indicates coupled, allosteric regions. In particular, the thermal energy tends to propagate along pathways used for allosteric control. As shown in Fig. 13, we find a direct pathway of thermal communication between our new pocket and the folate-binding site; the pathway involves Phe137, Ile155, Tyr111, and a crystallographically conserved water molecule (found in 1RX1, 1RX4, 1RX5, 1RX6, 1RA1, and 1RA3).

Additionally, we find further support in the MD. Several water molecules exhibit notable occupancies and lifetimes in interactions within the new site (see Appendix 1). The correlated dynamics of the closed-loop simulation shows distinct patterns. Ile155, Phe153, Phe137, and Tyr111 are all positively correlated with one another, and a clear, positive correlation is seen along the pathway from ATD, including the sites of all three of the β-bulge mutants. We next turned our attention to communication between the new site and the folate-binding site. Ile5, Ala6, Asp27, Phe31, Lys32, Arg52, Arg57, Ile94 and Tyr100 are within 3.5 Å of MTX in the 1RA3 crystal structure,[6] and several show correlations with residues in the new site. Ile5 and Ala6 are part of an island of strong positive correlation that involves Tyr111, part of the ATD pathway. Asp27 is part of a transient region of strong positively correlated dynamics with Glu139, His141, and Phe153 which are part of a turn between βG and βH. Ile94 and Tyr111 are positively correlated throughout the simulation.

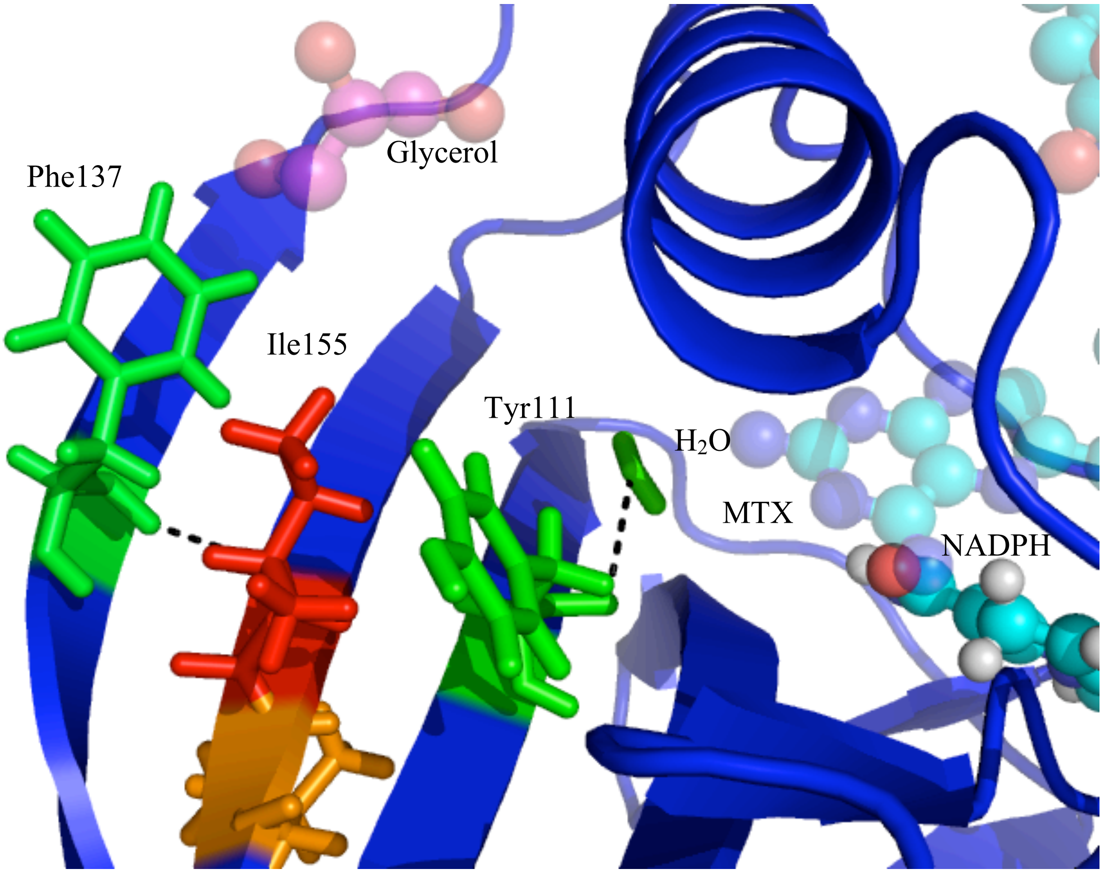


Figure 13. Pathway for communication between the new site and the folate-binding site. After 3ps of ATD, DHFR has been colored by calculated atomic fluctuations, where blue shows low motion and red shows the accelerated motion of the heated residue, Ile155. NADPH is shown in ball-and-stick. A clear path can be seen from Phe137 to Ile155 to Tyr111 to a conserved crystallographic water to the folate-binding site (indicated here by methotrexate (MTX)). The glycerol molecule from 1DF7 marks the entrance to the cavity, which runs across the top of the indicated residues from glycerol to MTX. The positions of the methotrexate seen in the 1RA3 crystal structure and the glycerol found in the 1DF7 crystal structure are shown in transparent ball-and-stick. Hydrogen bonds are shown in black dashes.

Conclusion and summary

We have shown that the DHFR•NADPH complex samples conformations relevant to the folate-bound state and catalysis, but more importantly, we provide new insights into the basic biophysics of the dynamics of DHFR. Previous MD and QM/MM work has shown that a network of coupled motions facilitates hydride transfer and that the corresponding correlated dynamics are present primarily in the reactant state. We have shown that these motions are present in DHFR•NADPH, pre-existing the binding of DHF. It is intriguing that the motions are present only *transiently* in DHFR•NADPH, and many are not seen on a standard multi-nanosecond timescale. Proteins are known to sample a conformational ensemble of states, only some of which are relevant to ligand binding and catalysis. Our results indicate that they sample a similar ensemble of correlated dynamics.

This finding has important implications for any system in which correlated dynamics affect binding and catalysis. Examining dynamics on an appropriate timescale is vitally important. The correlated dynamics in this study are transient, present well before ligand binding, and washed out when examined at timescales of several nanoseconds. We strongly encourage other researchers to examine dynamics on both long and short timescales relevant to their particular system.

Lastly, we have used a multitude of techniques to identify and characterize a new potential binding site, showing consistent patterns with diverse approaches. Taken together, this provides a good deal of support that the new pocket may be important as a potential allosteric or regulatory site. If this is true, there are many exciting implications, the most obvious of which is that small molecules binding in this site could form new classes of anti-cancer and anti-microbial drugs. This is further supported by the appearance of a crystallographic additive in the new site of a structure of *M. tuberculosis* DHFR.

Acknowledgments

We thank Dr. Warren DeLano for his assistance in writing and speeding up custom PyMOL scripts and Alex Buck, Dr. Bosco Ho, and Prof. David Agard for their help with the ATD simulations. Financial support from the National Institutes of Health (Grant GM65372), the Beckman Young Investigator Program and the University of Michigan's Molecular Biophysics Training Program (NIGMS grant GM008270) is gratefully acknowledged.

Supporting information available. Information about correlated dynamics of the loops and network of coupled motions is available in Appendix 1, as is information about the behavior of hydrogen bonds and bridging water molecules throughout the trajectories.

References

(1) Berg, J. M.; Tymoczko, J. L.; Stryer, L. *Biochemistry*; 5th ed.; W. H. Freeman and Company: New York, 2002.

(2) Schnell, J. R.; Dyson, H. J.; Wright, P. E. *Annu. Rev. Biophys. Biomol. Struct.* **2004**, *33*, 119-140.

(3) Radkiewicz, J. L.; Brooks, C. L., III *J. Am. Chem. Soc* **2000**, *122*, 225-231.

(4) Boehr, D. D.; McElheny, D.; Dyson, H. J.; Wright, P. E. *Science* **2006**, *313*, 1638.

(5) Venkitakrishnan, R. P.; Zaborowski, E.; McElheny, D.; Benkovic, S. J.; Dyson, H. J.; Wright, P. E. *Biochemistry* **2004**, *43*, 16046-16055.

(6) Sawaya, M. R.; Kraut, J. *Biochemistry* **1997**, *36*, 586-603.

(7) Khavrutskii, I. V.; Price, D. J.; Lee, J.; Brooks, C. L., III *Protein Sci.* **2007**, *16*, 1087-1100.

(8) McElheny, D.; Schnell, J. R.; Lansing, J. C.; Dyson, H. J.; Wright, P. E. *Proc. Natl. Acad. Sci. USA* **2005**, *102*, 5032-5037.

(9) Miller, G. P.; Benkovic, S. J. *Biochemistry* **1998**, *37*, 6336-6342.

(10) Osborne, M. J.; Schnell, J.; Benkovic, S. J.; Dyson, H. J.; Wright, P. E. *Biochemistry* **2001**, *40*, 9846–9859.

(11) Thorpe, I. F.; Brooks, C. L., III *Proteins: Struct. Funct. Bioinformatics* **2004**, *57*, 444-457.

(12) Rod, T. H.; Brooks, C. L., III *J. Am. Chem. Soc.* **2003**, *125*, 8718-8719.

(13) Fierke, C. A.; Johnson, K. A.; Benkovic, S. J. *Biochemistry* **1987**, *26*, 4085-4092.

(14) Osborne, M. J.; Venkitakrishnan, R. P.; Dyson, H. J.; Wright, P. E. *Protein Sci.* **2003**, *12*, 2230-2238.

(15) Adams, J.; Johnson, K.; Matthews, R.; Benkovic, S. J. *Biochemistry* **1989**, *28*, 6611-6618.

(16) Adams, J. A.; Fierke, C. A.; Benkovic, S. J. *Biochemistry* **1991**, *30*, 11046-11054.

(17) Ahrweiler, P. M.; Frieden, C. *Biochemistry* **1991**, *30*, 7801-7809.

(18) Cameron, C. E.; Benkovic, S. J. *Biochemistry* **1997**, *36*, 15792-15800.

(19) Chen, J. T.; Taira, K.; Tu, C. P. D.; Benkovic, S. J. *Biochemistry* **1987**, *26*, 4093-4100.

(20) Dion, A.; Linn, C. E.; Bradrick, T. D.; Georghiou, S.; Howell, E. E. *Biochemistry* **1993**, *32*, 3479-3487.

(21) Farnum, M. F.; Magde, D.; Howell, E. E.; Hirai, J. T.; Warren, M. S.; Grimsley, J. K.; Kraut, J. *Biochemistry* **1991**, *30*, 11567-11579.

(22) Fierke, C. A.; Benkovic, S. J. *Biochemistry* **1989**, *28*, 478-486.

(23) Miller, G. P.; Benkovic, S. J. *Biochemistry* **1998**, *37*, 6327-6335.

(24) Murphy, D. J.; Benkovic, S. J. *Biochemistry* **1989**, *28*, 3025-3031.

(25) Wagner, C. R.; Thillet, J.; Benkovic, S. J. *Biochemistry* **1992**, *31*, 7834-7840.

(26) Ohmae, E.; Ishimura, K.; Iwakura, M.; Gekko, K. *J. Biochem. (Tokyo)* **1998**, *123*, 839-846.

(27) Rajagopalan, P. T. R.; Lutz, S.; Benkovic, S. J. *Biochemistry* **2002**, *41*, 12618-12628.

(28) Rod, T. H.; Radkiewicz, J. L.; Brooks, C. L., III *Proc. Natl. Acad. Sci. USA* **2003**, *100*, 6980-6985.

(29) Wong, K. F.; Selzer, T.; Benkovic, S. J.; Hammes-Schiffer, S. *Proc. Natl. Acad. Sci. USA* **2005**, *102*, 6807-6812.

(30) Wong, K. F.; Watney, J. B.; Hammes-Schiffer, S. *J. Phys. Chem. B* **2004**, *108*, 12231-12241.

(31) Watney, J. B.; Hammes-Schiffer, S. *J. Phys. Chem. B* **2006**, *110*, 10130-10138.

(32) Lerner, M. G.; Bowman, A. L.; Carlson, H. A. *J. Chem. Inf. Model.* **2007**, *47*, 2358-2365.

(33) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. *Nucleic Acids Res.* **2000**, *28*, 235-242.

(34) DeLano, W. L. **2002**, Palo Alto, CA, USA, The PyMOL Molecular Graphics System 0.99rev8.

(35) Case, D. A.; Pearlman, D. A.; Caldwell, J. W.; Cheatham Iii, T. E.; Ross, W. S.; Simmerling, C. L.; Darden, T. A.; Merz, K. M.; Stanton, R. V.; Cheng, A. L. **1999**, University of California, San Francisco, AMBER 6.

(36) Cornell, W. D.; Cieplak, P.; Bayly, C. I.; Gould, I. R.; Merz, K. M.; Ferguson, D. M.; Spellmeyer, D. C.; Fox, T.; Caldwell, J. W.; Kollman, P. A. *J. Am. Chem. Soc.* **1995**, *117*, 5179-5197.

(37) Holmberg, N.; Ryde, U.; Bulow, L. *Protein Eng.* **1999**, *12*, 851-856.

(38) Jorgensen, W. L.; Chandrasekhar, J.; Madura, J. D.; Impey, R. W.; Klein, M. L. *J. Chem. Phys.* **1983**, *79*, 926-935.

(39) MOE **2005**, Chemical Computing Group, Montreal, Canada, MOE 2005.06.

(40) Case, D. A.; Darden, T. A.; Cheatham III, T. E.; Simmerling, C. L.; Wang, J.; Duke, R. E.; Luo, R.; Merz, K. M.; Wang, B.; Pearlman, D. A.; Crowley, M.; Brozell, S.; Tsui, V.; Gohlke, H.; Mongan, J.; Hornak, V.; Cui, G.; Beroza, P.; Schafmeister, C.; Caldwell, J. W.; Ross, W. S.; Kollman, P. A. **2004**, University of California, San Francisco, AMBER 8.

(41) Ota, N.; Agard, D. A. *J. Mol. Biol* **2005**, *351*, 345-354.

(42) Petřek M., O. M., Banáš P., Košinová P., Koča J. and Damborský J. *BMC Bioinformatics* **2006**, *7*.

(43) Bryliński, M.; Prymula, K.; Jurkowski, W.; Kochańczyk, M.; Stawowczyk, E.; Konieczny, L.; Roterman, I. *PLoS Comput. Biol.* **2007**, *3*, 0909-0923.

(44) Brylinski, M.; Konieczny, L.; Roterman, I. *Bioinformation* **2006**, *1*, 127-129.

(45) Edelsbrunner, H.; Facello, M.; Fu, P.; Liang, J. *System Sciences, 1995. Vol. V. Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences* **1995**, *5*.

(46) Epstein, D. M.; Benkovic, S. J.; Wright, P. E. *Biochemistry* **1995**, *34*, 11037-11048.

(47) Falzone, C. J.; Wright, P. E.; Benkovic, S. J. *Biochemistry* **1994**, *33*, 439-442.

(48) Li, L.; Wright, P. E.; Benkovic, S. J.; Falzone, C. J. *Biochemistry* **1992**, *31*, 7826-7833.

(49) Agarwal, P. K.; Billeter, S. R.; Rajagopalan, P. T. R.; Benkovic, S. J.; Hammes-Schiffer, S. *Proc. Natl. Acad. Sci. USA* **2002**, *99*, 2794-2799.

(50) Eisenmesser, E. Z.; Millet, O.; Labeikovsky, W.; Korzhnev, D. M.; Wolf-Watz, M.; Bosco, D. A.; Skalicky, J. J.; Kay, L. E.; Kern, D.; Contact, N. P. G. *Nature* **2005**, *438*, 117-121.

(51) Agarwal, P. K.; Billeter, S. R.; Hammes-Schiffer, S. *J. Phys. Chem. B* **2002**, *106*, 3283-3293.

(52) Pan, H.; Lee, J. C.; Hilser, V. J. *Proc. Natl. Acad. Sci. USA* **2000**, *97*, 12020-12025.

(53) Benkovic, S. J.; Hammes-Schiffer, S. *Science* **2003**, *301*, 1196-1202.

(54) Wang, L.; Tharp, S.; Selzer, T.; Benkovic, S. J.; Kohen, A. *Biochemistry* **2006**, *45*, 1383-1392.

(55) Hammes-Schiffer, S.; Benkovic, S. J. *Annu. Rev. Biochem.* **2006**, *75*, 519-541.

(56) Dion-Schultz, A.; Howell, E. E. *Protein Eng.* **1997**, *10*, 263-272.

(57) Soga, S.; Shirai, H.; Kobori, M.; Hirayama, N. *J. Chem. Inf. Model.* **2007**, *47*, 400-406.

(58) Howell, E. E.; Booth, C.; Farnum, M.; Kraut, J.; Warren, M. S. *Biochemistry* **1990**, *29*, 8561-8569.

(59) Brown, K. A.; Howell, E. E.; Kraut, J. *Proc. Natl. Acad. Sci. U. S. A.* **1993**, *90*, 11753-11756.

(60) Li, R.; Sirawaraporn, R.; Chitnumsub, P.; Sirawaraporn, W.; Wooden, J.; Athappilly, F.; Turley, S.; Hol, W. G. J. *J. Mol. Biol.* **2000**, *295*, 307-323.

Chapter 4

PyPAT: a python-based toolset to aid in the analysis of protein structures and trajectories

Introduction

While packages such as AMBER,[1] CHARMM,[2] and GROMACS[3] provide excellent tools for performing molecular dynamics (MD) simulations, there are some gaps in the tools provided by these packages for the analysis of those simulations. Other packages, such as MMTSB[4] and WHAT IF[5] help to fill in this gap. To this end, we have written PyPAT (Python-based Protein Analysis Tools), a suite of open-source analysis tools.

A wide variety of programming languages are available for consideration in the creation of such tools. Tools for the analysis of MD trajectories are, in general, significantly less computationally intensive than those used in the calculation of those trajectories. Therefore, the speed of executing the programs is less of a consideration than the ability to rapidly develop and maintain the code. With this in mind, we have chosen the scripting language Python, which is very easy to read, write and maintain, rather than a faster, compiled language (C, C++, FORTRAN, etc.). Python has an additional advantage: when execution speed is necessary, it is very easy to call code from libraries written in a faster, compiled language. As an example of this, we have incorporated numpy[6], an efficient, compiled package to perform more complex numerical calculations.

When implementing these tools, we found that several tasks were performed repeatedly. Whenever possible, these tasks have been abstracted into reusable libraries. These speed the development further tools by making it possible for end users to easily access these common functions instead of rewriting their own code to perform common, repetitive tasks.

MD calculations are typically performed via a command-line interface (CLI). Since our tools are used in the analysis of such trajectories, all of our tools are accessed via a CLI. Python provides several modules to aid in the creation of command-line tools. All of our tools make use of the optparse module, which provides both consistent, easy to understand documentation as well as reasonable default values whenever possible.

Our tools are primarily meant to interact with the AMBER suite of programs,[1] and have been extensively tested on both OS X and Linux. Many may be installed on Windows systems, but this is generally unsupported. A full list of dependencies is provided with the tools, including Python 2.5 (www.python.org), matplotlib 0.9.0 (matplotlib.sourceforge.net), numpy 1.0.1,[6] PyMOL 1.0,[7] gnuplot (www.gnuplot.info) and ImageMagick 6.3.2 (www.imagemagick.org). These are all freely available and are, with the exception of ImageMagick, entirely open-source. We feel that this reliance on open-source tools is particularly important. Access to source code is important for understanding and modifying code. We expect that researchers will want to modify and extend PyPAT to suit their purposes.

Software availability

Software is available free of charge for all users at http://www.umich.edu/~carlsonh and is released under an open-source license. Many utilities require input from ptraj, which is covered under the AMBER license.[1]

Tools

Graphical display of MD parameters over time

AMBER contains sander, a program that performs MD simulations. Analyzing MD simulations requires assessment of the convergence of dynamic properties, such as temperature, pressure, total energy, potential energy, kinetic energy and various error estimates. Sander outputs this information to a text file, and our script (`parse_sander_output.py`) was created to aid in visualizing and quickly assessing convergence. The script creates graphical images of the time-evolution of the properties. It also creates an html file that shows thumbnails of all these images, nested with links to larger versions. Our script parses the sander output file and creates individual tab-delimited files of temperature vs. time, potential energy vs. time, etc. Additionally, the data of all the dynamic properties is compiled into one large tab-delimited file. The tab-delimited format was chosen because it is easily read by any spreadsheet code, making it convenient for further analysis and creating figures.

Bridging water analysis

Bridging water molecules (BWs) are extremely important to the structure, dynamics and function of proteins and nucleic acids. They perform many roles, including stabilizing structures,[8-14] mediating hydrogen bonds,[8-10,12,14,15] regulating flexibility and motion,[8,11-13] altering local electrostatic properties,[8,10,14,15] mediating hydrophobic contacts,[9,10] and contributing directly to catalytic reactions.[8,10-12,14,15] They have important implications for drug design[9] and are particularly relevant in the study of nucleic acids[12,13] and protein-nucleic acid interactions.[14,15] MD simulations performed in explicit water provide a unique opportunity to examine bridging water interactions. The ptraj module of AMBER[1] calculates hydrogen bonds between a protein atom and water molecules, but not protein-water-protein bridges. To our knowledge, the analysis of BW is tedious for all MD packages, not just AMBER. It typically requires the user to search traditional hydrogen-bonding data to find individual waters that interact with more than one hydrogen-bonding group within a complex. This is straightforward when the user already knows to look for a BW (for instance, the DBLWAT command in WHAT IF can be used[5]). However, it is difficult to identify previously unknown interactions. This is unfortunate because insights like these are strength of MD. Our codes (`collect_water_bridges.py` and `display_bridging_interactions.py`) fill this gap in the currently available toolsets. In the default installation, it can provide information about water bridges between standard amino acids, nucleic acids, and NADPH. Users can easily extend it to provide information about other ligands, non-standard amino acids, etc. For convenience, we will refer to those molecules (proteins, nucleic acids, and ligands) interacting with BW as "protein" in the following discussion.

The code `collect_water_bridges.py` is used within PyMOL[7] to perform the first step in processing the trajectory files. For each frame in the trajectory, all water molecules with a heavy atom-heavy atom distance of ≤4.0 Å to the protein are analyzed for their interactions with hydrogen-bond donors and acceptors on the protein (the 4.0 Å default distance can be changed via an argument in the CLI). We chose to calculate the interactions using an overly generous cutoff in the first stage and display the data with more stringent cutoffs in the second stage. This allows the user to change criteria and examine the effects without re-processing the entire trajectory. Distance and angle information are accumulated in the first code with the final step recording any water molecule that has more than one interaction with the protein. The second code, `display_bridging_interactions.py`, inverts the data from a water-centric perspective into bridging interaction (BI) data with protein-water-protein triplets.

There are several unusual cases that can arise in BIs that complicate the analysis. BW can provide multiple BIs on occasion. For example, a BW may donate hydrogen bonds to two protein atoms while also accepting a hydrogen bond from a third protein atom (3 BIs). Suppose that after 200 ps, there is a shift, and two water molecules enter so that the same 3 BIs are made, but each by an individual BW. Before and after the shift, the same number of interactions is made, regardless of the number of BWs. We report both events as 3 BIs. Instead, if the shift at 200 ps causes the original BW to lose one of the interactions and no other water compensate, the number of BIs drops from 3 to 1.

The `display_bridging_interactions.py` code has several options. The user can select a second, more stringent cutoff for the heavy atom-heavy atom distances that define the hydrogen bonds. This can be any value less than or equal to the 4.0-Å cutoff in the first phase (if the user sets the first-phase cutoff smaller or greater than 4.0 Å, that will change the maximum distance available in this analysis). Output can be restricted to BIs involving a particular set of residues. It can be sorted by either occupancy or ALT. It can be filtered by minimum occupancies and ALTs. Users may specify a minimum required lifetime. For example, one may choose not to include interactions that last less than three frames of the trajectory. Eliminating these transient interactions may give a clearer picture of the occupancy and ALT of a particular BI.

We find that it is common to have BIs where the water drifts briefly beyond the distance cutoff and then comes back into range. In a situation where, for example, a particular water molecule interacts in one BI for 300 ps, drifts away for 1 ps, and comes back to the same BI for another 300 ps, it may be more appropriate to record one 601-ps interaction, rather than two 300-ps interactions. Thus, we allow the user to specify an additional "looseness" parameter that indicates how long that transient displacement can be. This is in keeping with analysis of lifetimes of standard hydrogen bonds by ptaj, except that the user can set the looseness parameter, while ptraj uses a set default of one frame for solvent interactions and zero for non-solvent interactions.

When calculating the occupancy of a BI, we must properly account for cases in which more than one BW occupies a particular BI at the same time. For instance, suppose that we have a trajectory with 10 snapshots, and a particular BI is occupied by two BWs in the first five snapshots and one BW in the last five. It is possible to double-count and report an occupancy of 150%. Although this is easy to interpret when the occupancy is greater than 100%, it is less clear when the occupancy is less than 100%. For instance, an occupancy of 80% could mean that one BW was making the interaction for eight snapshots or that two waters were making the interaction for four snapshots. For this reason, we do not double-count in these cases. Rather, occupancy is defined as the percentage of frames in the trajectory for which the BI is occupied, regardless of the number of BWs involved. If a BI is particularly important for a system, the user will surely visualize those interactions and determine the role of multiple BWs.

For each BI, `display_bridging_interactions.py` reports BI using a format similar to ptraj:

```
BI 900.0ps Occ:72.9% ALT:29.7ps #H2O:158
               Bridge:(Glu154 OE2) to (Ser135 HG): oxx**.xo*ox*@**@*x
```

In order, the format provides the following information:

1. The total time the BI is occupied, in picoseconds
2. The occupancy (percentage of the trajectory for which the BI is occupied)
3. The average lifetime (ALT – the average length of time for which a BI is occupied; longer ALTs may be relevant to the thermodynamic stability of BWs and BIs)
4. The total number of times the BI is occupied throughout the trajectory
5. The atoms and types of hydrogen-bonding interactions involved (above, the BW is donating to the side chain of Glu154 and accepting from the side chain of Ser135)
6. A graph of the occupancy throughout the MD (the trajectory is divided into user-defined sections – 18 in the example above – and each section is coded in keeping with ptraj's notation: "`@`" for a BI that was occupied during 95-100% of the section, "`*`" denotes 80-95%, "`o`" for 40%-60%, "`-`" for 20-40%, "`.`" for 5-20%, and a blank space for 0%-5%)
7. Optionally, a list of IDs of all of the BWs that occupy the interaction (not shown in the example above)

Hydrogen bonding

AMBER's ptraj module does an excellent job of calculating hydrogen bonds throughout MD simulations. Unfortunately, in examining our systems, we have found that we often run out of memory when using ptraj for moderately sized (≥10 ns) simulations. The solution to this problem is to use ptraj to calculate hydrogen bonds for subsections of the trajectory that do fit into memory and then use the script (`combine_hbonds.py`) to piece together the results to produce data from the trajectory as a whole. Regardless of the memory issues, a full calculation of all of the hydrogen bonds in a protein system produces too much information to be easily understood. At the command line, our script allows the user to sort the results based on several factors including total occupancy and residue ID, as well as allowing the user to filter the results based on a list of important residues and minimum required occupancy. The script also allows for a comparison of two (or more) trajectories. The output from corresponding hydrogen bonds is displayed together, allowing the user to see how hydrogen-bonding patterns differ in the trajectories. The combination of features in our script allows the user to calculate all of the hydrogen bonds first, and then filter it to display smaller, more understandable amounts of data from one or more trajectories. It is more efficient to calculate all of the interactions and display subsets than to recalculate for each individual case of interest.

Correlated dynamics

Correlated dynamics have been used to study relative motion in many protein systems,[16-21] and we use a representative protein with a bound cofactor in our examples below. Given a vector X whose components are random variables ($x_1, \ldots, x_n$) with means ($\mu_1, \ldots, \mu_j$) and standard deviations ($\sigma_I, \ldots, \sigma_j$), the covariance of two entries, x_i and x_j is defined as the expectation value

$$\mathrm{Cov}(x_i,x_j) = < (x_i - \mu_i)(x_j - \mu_j) >.$$

The magnitude of the covariance depends on the standard deviations, and is therefore often scaled to obtain the correlation:

$$\mathrm{Corr}(x_i,x_j) = \mathrm{cov}(x_i,x_j) / (\sigma_i\sigma_j).$$

The correlation matrix is formed in the same way as the covariance matrix, with values normalized to range from -1 to 1. Two completely synchronized elements have a correlation of 1, two perfectly anti-correlated elements have a correlation of -1, and two independent variables have a correlation of 0.

We have written scripts that automate the calculation of correlation matrices from an MD trajectory. These scripts use AMBER's ptraj module to calculate correlation matrices, allowing the user to optionally align the trajectory and filter out hydrogens and other atoms. In our work, we have found that analyzing correlations in long trajectories requires special treatment. It is important to look at dynamics at various timescales, rather than simply averaging them over the entire trajectory. Our scripts allow the user to calculate the correlation matrices in windows of arbitrary length and spacing throughout the trajectory. For instance, we prefer to calculate these matrices in windows spanning 1 ns; we step this 1-ns window through the entire trajectory at 100 ps increments (e.g., 1.5-2.5 ns, 1.6-2.6 ns, 1.7-2.7 ns, etc.). Of course, a user can calculate the correlation over the entire trajectory by simply setting that as the length of the window.

One of the biggest advancements is the collection of scripts we have written to aid in the visualization of correlated dynamics. First, several different color schemes (technically known as color maps) are typically used in the literature (e.g., see the difference between figures in representative publications[16,18,21]). Matplotlib (our chosen plotting package) has several standard color maps and allows for the creation of user-defined color maps. In addition to supporting the standard color schemes via the command line, we have created a custom color map that mimics that of Radkiewicz and Brooks.[16] In that coloring scheme, strongly anti-correlated regions are dark blue; yellow and red are used for medium and strong positive correlations, and light blue is used for uncorrelated regions. In the event that a different coloring scheme is desired, users can define their own color maps.

Most investigations of correlated dynamics in the literature focus on correlations between α-carbons. We find that significant information is contained in the correlations of other atoms in the system. Thus, our scripts provide the option to calculate α-carbon, main-chain heavy-atom, or all-atom correlations in ptraj and display the information in a variety of ways. A comparison may be seen in Fig. 14. We typically find that main-chain, heavy-atom plots are the most informative. It is worth noting that, by default, the main-chain and all-atom figures include correlations with ligands and cofactors. These are extremely important, but have typically been ignored, because of the exclusive use of Cα analysis in the literature. All-atom plots contain the full information, but are often less useful in practice. Side-chain atoms, even those in regions of rigid structure, tend to be uncorrelated with other atoms. Furthermore, the common use of SHAKE[1,22] will remove correlation information for hydrogens. Our tools also allow the user to calculate and plot the following values on a per-residue basis: average, maximum, minimum, and greatest absolute value. Each of these quantities provides useful information. For example, looking at the minimum plots allows a researcher to focus specifically on anti-correlated motion. The greatest-absolute-value plots have the advantage that regions of correlated and anti-correlated motion stand out more sharply from the background. An example is shown in Fig. 14c.

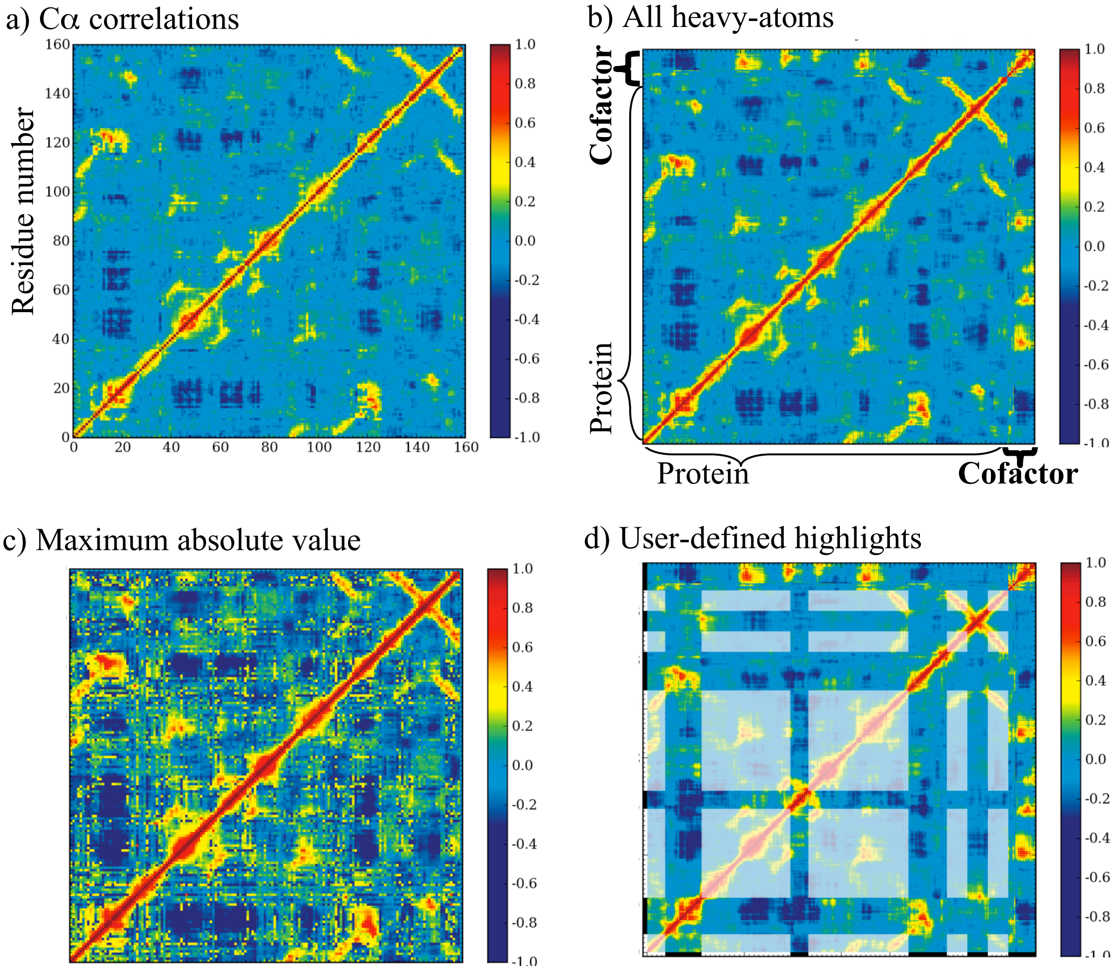


Figure 14. Correlated dynamics plots. These plots show the correlated dynamics from 1 ns of an MD simulation performed on a representative protein with a bound cofactor. a) A standard plot of α-carbon correlations. The numbering on the axes denotes the residue involved. The color scheme is shown on the right. b) The same data plotted for all main-chain heavy atoms and the associated cofactor. The correlated and anti-correlated regions are more detailed, and the important interactions with the cofactor are revealed on the top and right sides of the figure. c) For each residue, the largest absolute value of its atoms is plotted. Regions of positive and negative correlations may be more easily detected in this coloring scheme. d) The main-chain, heavy-atom plot is shown with a user-defined highlight of specific regions. This can be used to filter out information in other regions. The reader should be aware that some of these plots contain more information than can be seen at the resolution printed on this page.

These plots can contain a seemingly overwhelming amount of information. Thus, our scripts provide several options for highlighting areas of interest. This is done by "fading out" the alternative regions. In Fig. 14d we show correlated dynamics, with user-defined regions highlighted. Focusing on different regions, like the intersection of a binding site and a regulatory site, can provide new insight into the structure and dynamics of system.

Finally, we have found that the correlated dynamics can change throughout the course of an MD simulation. Our scripts make movies of the correlated dynamics, allowing the user to visualize their changes with time. This is only possible if the user chooses to calculate the correlations in windows smaller than the full length of the trajectory. In order to make this visualization as easy as possible, html pages are generated that contain small versions of the movies, linking to the full-sized versions.

Conclusion

We have created several scripts to augment the analysis of AMBER trajectories, building on the existing functionalities of ptraj. These tools are written in Python and released under an open-source license in the hopes that other researchers will be able to easily use and extend them. In particular, by changing the input format, one could use the scripts with different packages such as CHARMM and GROMACS.

Acknowledgements

I thank Dr. Steve Spronk for his help throughout this project. I thank Dr. Warren DeLano for his assistance in writing and speeding up custom PyMOL scripts. I also thank Mark Ditzler and Dr. Nils Walter for the definitions of RNA hydrogen-bond donors and acceptors. Financial support from the National Institutes of Health (Grant GM65372), the

Beckman Young Investigator Program and the University of Michigan's Molecular Biophysics Training Program (NIGMS grant GM008270) is gratefully acknowledged.

Supporting information. The source code for PyPAT is available in Appendix 2.

References

(1) Case, D. A.; Darden, T. A.; Cheatham III, T. E.; Simmerling, C. L.; Wang, J.; Duke, R. E.; Luo, R.; Merz, K. M.; Wang, B.; Pearlman, D. A.; Crowley, M.; Brozell, S.; Tsui, V.; Gohlke, H.; Mongan, J.; Hornak, V.; Cui, G.; Beroza, P.; Schafmeister, C.; Caldwell, J. W.; Ross, W. S.; Kollman, P. A. **2004**, University of California, San Francisco, San Francisco, AMBER 8.

(2) Brooks, B. R.; Bruccoleri, R. E.; Olafson, B. D.; States, D. J.; Swaminathan, S.; Karplus, M. *J. Comput. Chem* **1983**, *4*, 187-217.

(3) Spoel, D. V. D.; Lindahl, E.; Hess, B.; Groenhof, G.; Mark, A. E.; Berendsen, H. J. C. *J. Comput. Chem.* **2005**, *26*, 1701-1718.

(4) Feig, M.; Karanicolas, J.; Brooks, C. L., III *J. Mol. Graph. Model* **2004**, *22*, 377-395.

(5) Vriend, G. *J. Mol. Graphics* **1990**, *8*, 52-56.

(6) Oliphant, T. E. *Guide to NumPy*; Trelgol, 2006.

(7) DeLano, W. L. **2002**, Palo Alto, CA, USA, The PyMOL Molecular Graphics System 0.99rev8.

(8) Nakasako, M. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* **2004**, *359*, 1191-1206.

(9) Plumridge, T. H.; Waigh, R. D. *J. Pharm. Pharmacol.* **2002**, *54*, 1155-1179.

(10) Saenger, W. *Annu. Rev. Bioph. Biom.* **1987**, *16*, 93-114.

(11) Smith, J. C.; Merzel, F.; Bondar, A. N.; Tournier, A.; Fischer, S. *Philosophical Transactions: Biological Sciences* **2004**, *359*, 1181-1190.

(12) Westhof, E. *Annu. Rev. Bioph. Biom.* **1988**, *17*, 125-144.

(13) Feig, M.; Pettitt, B. M. *Structure* **1998**, *6*, 1351-4.

(14) Jayaram, B.; Jain, T. *Annu. Rev. Biophys. Biomol. Struct.* **2004**, *33*, 343-361.

(15) Bergqvist, S.; Williams, M. A.; O'Brien, R.; Ladbury, J. E. *J. Mol. Biol.* **2004**, *336*, 829-842.

(16) Radkiewicz, J. L.; Brooks, C. L., III *J. Am. Chem. Soc* **2000**, *122*, 225-231.

(17) Rod, T. H.; Radkiewicz, J. L.; Brooks, C. L., III *Proc. Natl. Acad. Sci. USA* **2003**, *100*, 6980-6985.

(18) Thorpe, I. F.; Brooks, C. L., III *Proteins Struct. Funct. Bioinformat.* **2004**, *57*, 444-457.

(19) Watney, J. B.; Hammes-Schiffer, S. *J. Phys. Chem. B* **2006**, *110*, 10130-10138.

(20) Wong, K. F.; Selzer, T.; Benkovic, S. J.; Hammes-Schiffer, S. *Proc. Natl. Acad. Sci. USA* **2005**, *102*, 6807-6812.

(21) Wong, K. F.; Watney, J. B.; Hammes-Schiffer, S. *J. Phys. Chem. B* **2004**, *108*, 12231-12241.

(22) Ryckaert, J.-P.; Ciccotti, G.; Berendsen, H. J. C. *J. Comput. Phys.* **1977**, *23*, 327-341.

Chapter 5

Automated clustering of probe molecules from solvent mapping of protein surfaces

Introduction

In an effort to understand protein binding and function, researchers will often create a reciprocal map of a protein surface. Multiple-copy methods (MCM) use probe molecules to define these complementarity maps. These techniques flood the protein surface with hundreds of small molecule probes. The probes are then simultaneously and independently minimized to the protein's potential energy surface. Different probe molecules map out hydrophobic regions, hydrogen-bonding interactions, ion pairing, etc. Clusters of probes on the protein surface can define the most important among these interactions. However, grouping probes into clusters is not always straightforward, and yet, it is essential to mapping "hot spots"[1] and fragment-based drug design.[2-4] Despite this importance, there is little diversity in the methods used for defining clusters.

Clustering techniques

The most widely known MCM is multiple copy simultaneous search (MCSS).[5,6] The applications tend to remove probes throughout the minimization process.[5] Root-mean-square difference (RMSD)-based clustering is used at each step, and only the lowest-energy member of each cluster is retained. Additionally, an energy cutoff is used so that high-energy probes are removed throughout the minimization. In some implementations, these clusters are then ranked via energy-based techniques.[7]

MCSS2PTS automates the procedure of using the MCSS method to generate pharmacophore models.[6] In doing so, it uses standard RMSD methods to cluster the minimized probes.

Many different methodologies have been used to group molecules into clusters in chemical space such as RMSD-based methods, K-means clustering[8] and Jarvis-Patrick (JP) clustering,[9] but almost all MCSS-style methods have used standard RMSD techniques to cluster the probes in physical space.[5,6,10-13] This technique takes two forms, which we will refer to as seeded RMSD (sRMSD) and greedy RMSD (gRMSD) clustering. The sRMSD-clustering technique works as follows:

1. Choose a distance cutoff, R_{max}.
2. Choose any element, *i*, called the seed (typically, one begins with the lowest-energy element).
3. Assign all elements that are within R_{max} of *i* to a cluster and remove them from the list of elements.
4. Repeat steps 2-3 until the list is empty.

It is worth noting that mapping techniques that find a minimum energy probe and then force subsequent probes to remain outside a given RMSD cutoff effectively use a type of sRMSD clustering.

sRMSD is limited by the need to choose proper seeds. An improved approach, gRMSD clustering (also known as single-linkage clustering) works similarly but overcomes errors in poor choices for seeds:

1. Cluster elements as per sRMSD clustering.

2. If any element in one cluster is R_{max} of any element in another cluster, combine the two clusters.
3. Repeat step 2 until no more clusters can be combined.

An alternative, energy-based clustering technique has been implemented with MCSS.[14,15] Although the full details of their algorithm have not been published, a basic description has been given. Probes are clustered together when they have a similar set of van der Waals contacts with the protein. Thus, each cluster is identified by a "cluster signature" listing the amino acids with which that cluster interacts. These cluster signatures typically contain between three and thirteen residues.

One notable exception to the usage of sRMSD and gRMSD techniques is SitePrint.[16] SitePrint floods a structure with small chemical probes, which are then minimized to the protein surface. The k-mediod algorithm[17] is then used to cluster the probes. For k-mediod clustering, the user pre-determines a desired number of clusters (k). This number of clusters are then selected, and a representative element is chosen for each cluster. For each cluster, the sum of the dissimilarities between the individual elements and its representative element is calculated. The clusters and representative elements are chosen in a way that minimizes the total of these sums. The representative elements are then used to define pharmacophore models.

We have chosen not to use k-medioid or k-means clustering in this work because of their need to pre-define the number of interactions in a cavity, which is not easily extended across all systems. Instead, we compare gRMSD to a new Jarvis-Patrick (JP)[9] method. Based on two user-defined parameters (J and K_{min}, both positive integers), JP clustering is defined as follows:

1. Make a list of each element's J nearest neighbors.
2. Two elements cluster together if
   a. they are in each other's nearest neighbor lists and
   b. they have at least K_{min} of their J nearest neighbors in common.

We have also found it useful to add distance constraint to focus the neighbor list. A colloquial example of JP clustering is defining two people to be in the same social circle if they are alike enough and have enough similar friends in common. JP clustering is typically applied to questions of chemical similarity, where it has been shown to be one of the best-performing algorithms.[18] There is one example of symmetry-corrected JP clustering applied to physical similarity[19,20] where the authors used JP clustering to provide a conformational breakdown of six-membered ring compounds in the Cambridge Structural Database.[21] They found that JP clustering performed excellently. We have found no previous applications of JP clustering (symmetry-corrected or not) to MCM or to the study of protein-protein interfaces.

Flooding and minimization

Our previous work using MCM[22-25] has used the probe-placement routines from an older version of BOSS.[26] These routines were developed to simulate liquid-phase environments. Probes are initially placed in a regularly spaced grid such that all have the same orientation, and none can overlap. In this work, we investigate alternate flooding procedures, allowing for arbitrary probe density, as well as random initial coordinates and orientations.

Test systems

The types of interactions involved in protein-protein interfaces are well known and include hydrophobic, hydrogen-bonding, and ionic interactions as well as disulfide bridges.[27] These interactions are similar to those involved in protein-ligand interactions. Here, we apply the MUSIC (Multi-Unit Search for Interacting Conformers)[13] routine in BOSS to the study of protein-protein interfaces. We begin with a bound crystal structure. The two halves of the interface are separated, prepared, and flooded with probes, which are then minimized while the probe-probe interactions are ignored. The minimized positions of the probes are then defined as "clusters" using gRMSD and JP. Either method is considered successful if it defines clusters on one side of the interface that match chemical features on the opposite binding partner. It is important to note that this is not an evaluation of the ability of MCM to map protein surfaces; the locations of minimized probes are the same in both gRMSD and JP clusters.

There has been much interest in the study of protein-protein interfaces, particularly in the feasibility of drugs targeting protein-protein interfaces.[28,29] There have also been several efforts to classify and catalog different types of protein-protein interfaces,[30-33] as well as a visual survey of 136 homodimeric proteins.[34] In general, protein-protein interfaces have a modular architecture, composed of distinct, mostly independent clusters of interacting residues.[35] The contacts between the two sides of an interface are, for the most part, very complementary,[27] often involving "hot spots"[36]. It is sometimes possible to predict the structure of a protein-protein complex from the structure of the unbound component proteins, especially when the component proteins do not undergo significant conformational changes upon binding, as evidenced by the CAPRI competition.[37]

Our study was performed on seven biologically relevant protein-protein systems: CheY-CheA,[38] Thrombin-BPTI, Barnase-Barstar, hMms2-hUbc13, TRAF6-RANK, CDK6-p16^{INK4a}, and HDM2-p53. These systems were chosen to represent diverse biochemical systems and diverse types of protein-protein interfaces. All the systems studied had available crystal structures, allowing direct comparisons and validation of our results. By starting with a crystal structure of the bound interface, we reduce the need to consider protein flexibility, which can play a significant role in the formation of protein-protein complexes.[39]

Methods

Protein-protein interface selection and structure preparation

We obtained the complexes from the Protein Data Bank (PDB)[40] and used MolProbity[41] to ensure that side chains were properly oriented. MolProbidy also ensured that histidine residues had the proper protonation state. A PyMOL[42] script was employed to further investigate the hydrogen-bonding and steric interactions for all potential side-chain alterations suggested by MolProbity. All but one of the side-chain flips recommend by MolProbity were outside the protein-protein interfaces. All were deemed reasonable by visual inspection, and all were accepted. Any crystal-structure hydrogens were removed to ensure equivalent setup across all systems; this was reasonable given that the resolution ranged from 1.85-3.4 Å for the test systems. Once the residue conformations and protonation states were verified, the xleap module in AMBER[43] was used to add hydrogens to the protein structure. The sander_classic module in AMBER was used to minimize the hydrogens (heavy atoms fixed) by conjugate gradient minimization (until the either energy change of 1.0E-4 kcal/mol or and 10,000 steps were reached). The structures were split into the two separate halves of the protein-protein interface.

PDB ID 1YCR[44] is the human MDM2-p53 system, an important oncoprotein-tumor suppressor system. The crystal structure contains only two chains, A (HDM2) and B (p53).

PDB ID 1EAY[45] is the CheY-CheA system. The crystal structure contains two heterodimers, chains A/C and chains B/D, both of which exhibit slightly different binding modes. A number of residues were not resolved in the crystal structure. While none of these residues were directly involved in the protein-protein interface, the unresolved residues were closer to the interface in the A/C dimer, so we chose to investigate the B/D complex (chain B is CheY; chain D is CheA).

PDB ID 1BTH[46] is the thrombin-BPTI (bovine pancreatic trypsin inhibitor) system. The crystal structure contains two complexes, chains J/K/Q and chains H/L/P, each of which correspond to two thrombin chains and one BPTI chain. We have chosen to study the HL/P structure, since it contained fewer steric clashes identified by MolProbity.

PDB ID 1B27[47] is the Barnase-Barstar system. The crystal structure contains three dimers, chains A/D, B/E, and C/F. As per the original paper, we focus on the A/D dimer.

PDB ID 1J7D[48] is the human ubiquitin conjugating enzyme complex hMms2-hUbc13 system. The crystal structure contains only one complex, chain A (hMms2) and chain B (hUbc13).

PDB ID 1LB5[49] is the TRAF6-RANK system. The crystal structure contains only two chains, A (TRAF6) and B (RANK).

PDB ID 1BI7[50] is the cyclin-dependent kinase 6 (CDK6)-tumor suppressor p16^{INK4a} system. The crystal structure contains only two chains, A (CDK6) and B (p16^{INK4a}).

Probe selection

We use the following small-molecule probes, but our methods are easily generalized to other probes. Methanol is used to probe for hydrogen-bonding interactions. Methylammonium and acetate are used to map salt-bridge interactions and charged hydrogen-bonding interfaces. Ethane is used to probe for hydrophobic interactions. Benzene is used to probe for aromatic and hydrophobic interactions. It is worth noting that the benzene probes often pick out cation-π interactions, which can be particularly important in protein-protein interfaces.[51]

Probe flooding

We have developed an easy-to-use PyMOL[42] "wizard" that allows the user to flood the protein with an arbitrary number of probes, each of which is placed with a random position and orientation near the protein surface. The user places a sphere to define the active site via the PyMOL graphical user interface. The user selects the type of probe molecule to use, the number of probes to place in the active site, and the minimum distance allowed between a probe and the protein ("overlap distance"). The probes are then placed at random positions and orientations within the active site, subject to the constraint that they do not fall within the overlap distance of the protein.

We have implemented two methods for placing the probes within the sphere. The first method is the most straightforward. Cartesian limits are determined for the user-defined active-site sphere (x_{max}, x_{min}, y_{max}, y_{min}, z_{max}, z_{min}), and each probe is placed at a random coordinate within these limits. Since the limits are in Cartesian space and technically describe a cube around the active site, any probe that is placed outside the active-site sphere is rejected. This ensures a uniform probe distribution within the active site. Probes are then rotated a random degree around the x, y, and z axes in order to ensure a uniform sampling of orientations. At this point, probes that fall within the overlap distance of the protein are rejected. This procedure is simple to implement and can easily be extended to arbitrary shapes.

There are some systems for which the geometry dictates a need to bias the placement of probes towards the center of the active site, such as a deep narrow cleft where probes will become trapped in local minima on the surface and not map essential interactions at the bottom of the cleft. We have implemented a second method that biases sampling towards the center of the sphere, while retaining a uniform angular distribution. This is accomplished by contracting the uniform sampling in the radial direction. Although both methods are implemented, we have relied on the first one for this work.

The placement procedure is repeated until the desired number of probes has been placed within the active site. A series of Python scripts are then used to translate the flooded protein structures into appropriate BOSS[26] input and configuration files, at which point the probes are minimized via BOSS.

The centers and radii of the flooding regions were chosen separately for each system. In all cases, the flooding region was chosen to be large enough to encompass all relevant protein residues. For the protein-protein interfaces, the flooding region for one side was chosen to be large enough to encompass all relevant protein residues on both sides.

Probe minimization

We minimize the probe molecules onto the protein surface using BOSS 4.2[26] and the OPLS all-atom force field.[52] The MUSIC routine is implemented in BOSS by defining the probes as solvent and setting the solvent-solvent interactions to zero.[13] A low-temperature Monte Carlo search is used to perform a simultaneous random-walk minimization of all the probes. The resulting output is a coordinate file of all the probes, overlapping within local minima on the protein surface. This output was classified into clusters using both gRMSD and JP.

The distance between two probes

It is important to take symmetry of the probe molecule into account so that the arbitrary ordering of the atoms does not affect the comparison (see Appendix 1). For simplicity, we have tried to exclude hydrogens when possible. For instance, comparisons of ethanes and benzenes focus solely on the carbon atoms. We were not able to ignore hydrogens in one specific case, methanol, as the definition of hydrogen-bond donors and acceptors involved by both hydrogen atoms. The RMSD-based clustering used the carbons for ethane and benzene, the hydroxyl atoms of methanol, the carbon and nitrogen for methylammonium, and the carbons and oxygens of acetate.

Clustering the probes

We have implemented the gRMSD and JP techniques described above. As an enhancement to standard JP clustering, we allow the user to choose a maximum RMSD, R_{max}. Elements that are further than R_{max} from a given probe will not be listed in that probe's nearest neighbor list. This allows us to restrict the "looseness" of the clustering so that clusters are only comprised of nearby elements. It is worth noting that R_{max} only affects the elements in any given neighbor list; the clusters themselves may easily span distances much larger than R_{max}. The clustering algorithms are implemented in a series of object-oriented Python scripts, using PyMOL[42] as a front end. For our comparison of clustering techniques, we have only evaluated the probe clusters within 3.0 Å of the opposing protein face. For both gRMSD and JP, clusters are required to have a minimum of eight probes which is in keeping with our previous use of MCM in structure-based drug design (SBDD).

Results and discussion

Flooding

The advantages of the new flooding procedure can clearly be seen in Fig. 15, where the active site of HIV-1 protease is chosen as an example system. Fig. 15a shows the original method with 1253 benzene probes, and Fig. 15b shows the improved method with only 500 benzene probes. The new flooding procedure samples more of the active site by focusing the probes into the most important region. With more densely packed probes, we are more likely to sample partially occluded - but functionally important - interactions. We gain a significant speed advantage from being able to accurately map with fewer probes.

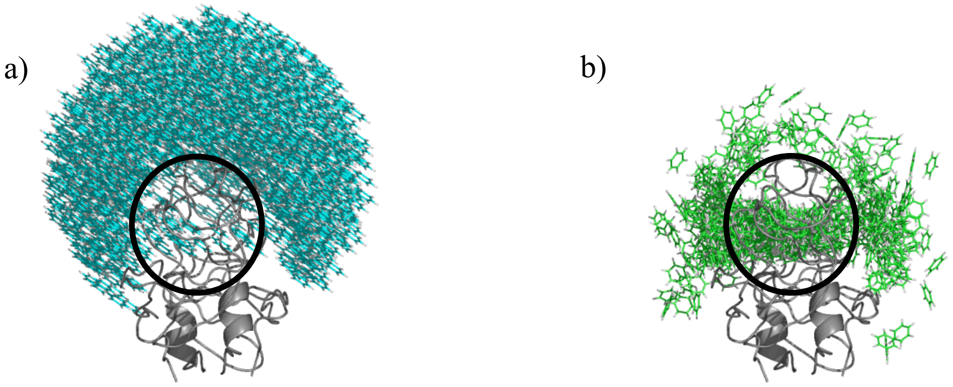


Figure 15. Improved flooding procedures. a) the active site of HIV-1 Protease (PDB ID 1HHP) is flooded with 1253 benzene molecules via the old procedure on the left and b) with 500 benzene molecules via the new procedure on the right. The new flooding procedure places more probes directly in the active site (circled in black).

Clustering

We have examined several automated clustering techniques. We find that the methods currently available in the literature are unsatisfactory for our purposes. Choosing clusters by hand produces the correct results but is extremely time-consuming. It is also a subjective process in which two people can easily come to different conclusions about what exactly constitutes a cluster. sRMSD clustering suffers from the fact that it is order dependent: choosing different "seeds" will produce different resulting clusters. gRMSD clustering remedies this defect. Indeed, we find that it can properly define any particular interaction for any particular system. However, its parameters must be tuned for each protein system because it is not able to simultaneously recognize loose and tight clusters.

We are thus forced to turn to more complex clustering algorithms. k-medioid clustering is able to simultaneously identify loose and tight clusters. However, the user is required to specify k, the desired number of clusters, *a priori*. Without examining the individual system, there is no way to pre-determine this parameter. Another popular technique, k-means clustering, suffers from the same defect. There is a large and significant body of literature on the subject of clustering.[17,53] We have examined several techniques and found that JP clustering is the simplest method that is able to accurately identify clusters of probes which properly map complementary interactions on a protein surface. JP clustering is both fast (after generating the list of J nearest neighbors, it can be completed in linear time) and deterministic. It should be noted that JP clustering with a reasonable R_{max}, a large value of J, and a K_{min} value of 1 will mimic RMSD clustering, but it will be much faster due to the fact that the JP clustering searches through a truncated neighbor list, rather than through all neighbors.

Fig. 16 shows an example with ethane probes. There are three clusters that we want to identify, one loose cluster in the upper right and two tighter ones on the left. If we use gRMSD and set the RMSD cutoff large enough to recognize the looser cluster, it joins the two tight clusters into one cluster. If we make the cutoff small enough to split these clusters, the looser cluster is not recognized as a single group. Rather, it is either recognized as three small clusters, or not at all, depending on the particular RMSD cutoff. JP clustering allows us to recover both tightly-packed clusters and looser clusters. The optimal JP parameters may be found in Table 1.

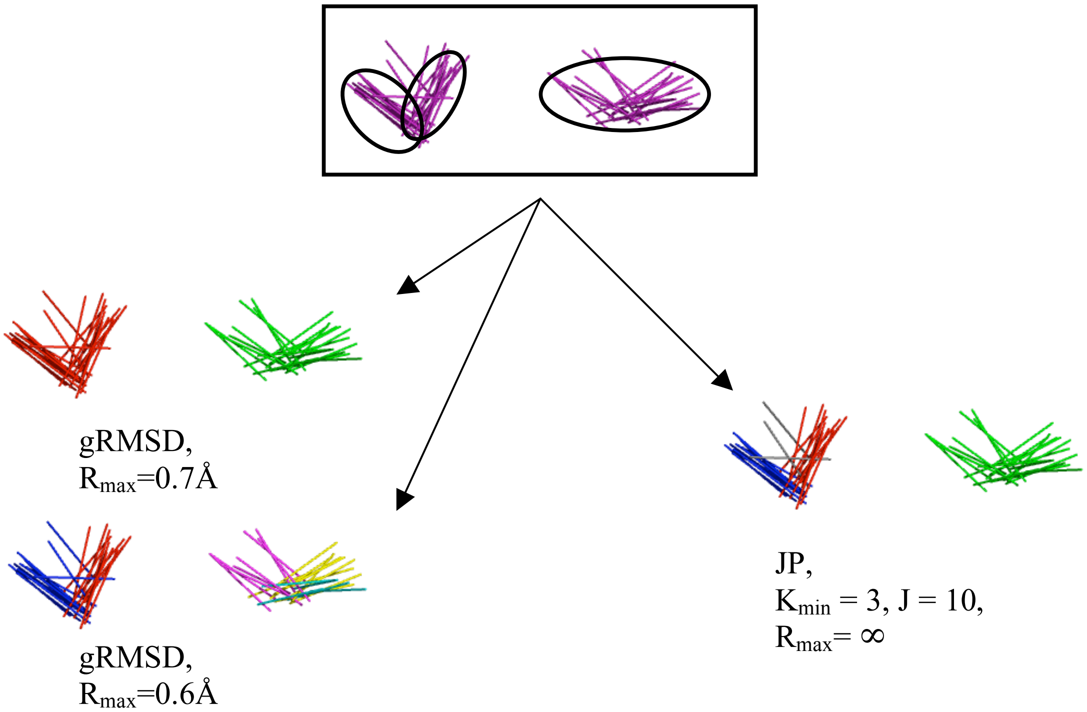


Figure 16. gRMSD clustering vs. JP clustering of ethane molecules. Hydrogens not shown for clarity. The top image shows three clusters from MCM minimizations, two closely spaced but densely packed clusters and one larger diffuse cluster that spans approximately the same dimensions. gRMSD clustering (left) is too coarse, identifying too few clusters (left-top) or too many (left-bottom) with a minimal change of R_{max}. JP clustering (right) is able to correctly discriminate between the three clusters. Clusters are identified by color: the purple ethanes in the top image are unclustered. Red, green, blue, purple, yellow, and teal represent different clusters in the lower images. Note that the JP method does not designate the few grey molecules in the lower right image as being part of any cluster.

Table 1. Optimal clustering parameters for ethane, benzene, methanol, acetate, and methylammonium. Ranges examined are shown.

	Ethane	*Benzene*	*Methanol*	*Acetate*	*Methylammonium*
J *(Range examined)*	25 (15-25)	25 (15-25)	17 (6-17)	15 (10-20)	25 (15-30)
K_{min} *(Range examined)*	5 (2-6)	4 (3-6)	2 (1-6)	7 (3-8)	5 (3-6)
R_{max} (Å) *(Range examined)*	1.6 Å (0.75-1.75)	1.1 (0.75-1.75)	1.4 (0.5-1. 5)	2.25 (0.75-2.5)	1.25 (0.75-1.5)

The example shown in Fig. 16 is by no means unique. In Fig. 17, we present the p53-HDM2 complex. Fig. 17a shows solvent mapping with methanol, as clustered by JP algorithms. This system was chosen because it demonstrates the superiority of JP clustering and also demonstrates the fact that some mapped sites are appropriate but not necessarily complemented by a binding partner. For instance, the purple and orange sites in Fig. 17b represent hydrogen-bonding interactions with backbone carbonyl oxygens (Val93 and Gln72). These are not complemented by p53, but can be successfully used in SBDD.[54] In Fig. 17b, six of the cluster sites directly map to interactions from p53: the black and blue clusters represent the interaction from Glu17, the yellow cluster represents a backbone interaction from Phe19, the pink cluster represents an interaction from Trp23, the green cluster represents a backbone carbonyl interaction from Leu25, and the cyan cluster represents an interaction from the C-terminal carboxylate of the peptide used in the crystal structure. The brown cluster represents an acceptor interaction with the solvent-exposed Lys51 side chain on the edge of the binding cleft. The cluster falls between the Lys51 of HDM2 and the Glu28 of the p53 helix, a position appropriate for a bridging water molecule (no waters were resolved in the original crystal structure).

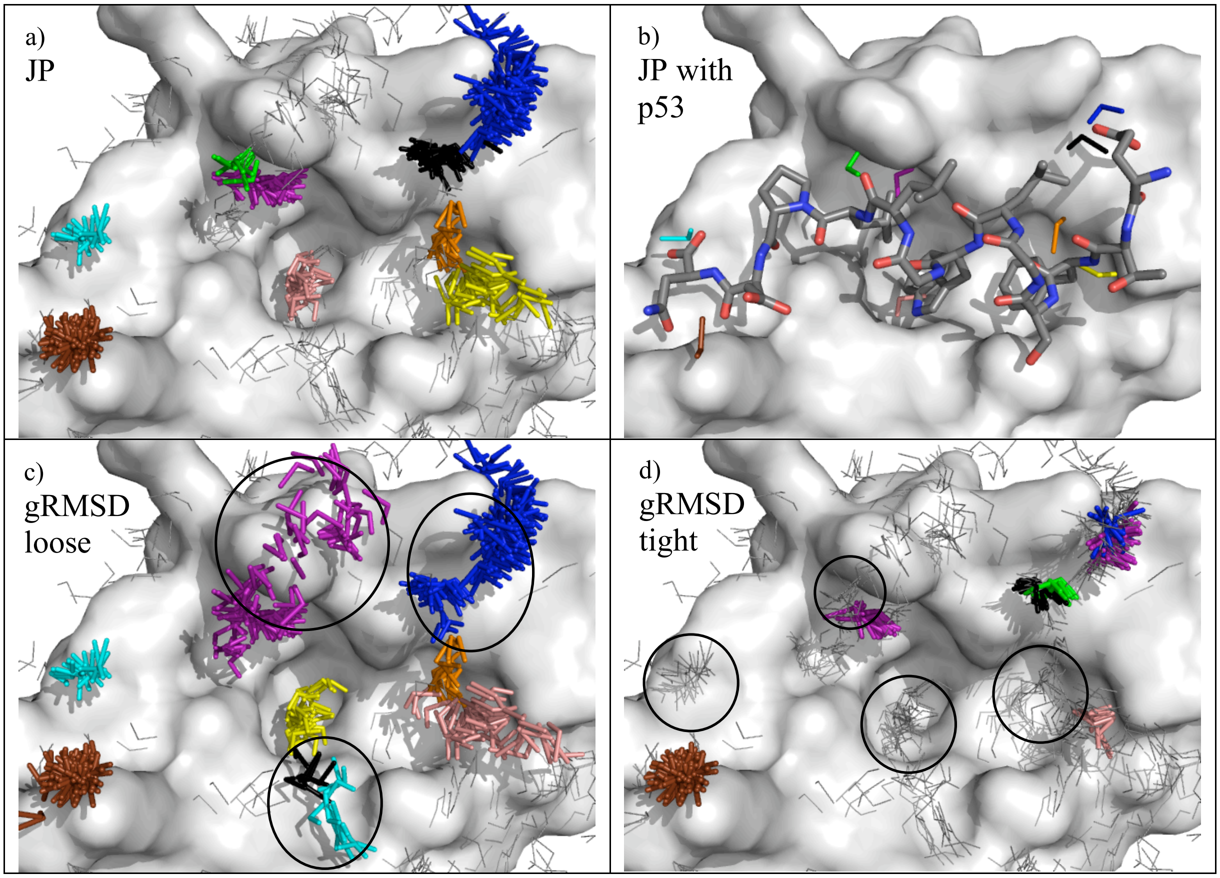


Figure 17. A comparison of JP and gRMSD clustering. Part a) shows the methanol JP clusters for HDM2. In part b), p53 has been overlaid to show its overlap with appropriate clusters. For clarity, only one element from each cluster is shown. Part c) shows the analogous gRMSD clustering at 1.4 Å where we see the typical results of clusters that are much too loose and large (circled); appropriate clusters have been joined together (upper right) and diffuse collections of probes have been marked as either new clusters (bottom center) or parts of other clusters (upper left). Part d) reducing the gRMSD cutoff to 0.5 Å eliminates this problem, but also eliminates important clusters (circled) and splits some clusters (upper right). The p53/HDM2 systems is shown as sticks/surfaces with hydrogens hidden for clarity. Methanol clusters within 3.0 Å of both protein surfaces and containing at least 7 elements are shown in colored sticks. Other methanols are shown in grey lines. For purposes of clarity, some probes outside the binding cleft are not shown, and aliphatic methanol hydrogens are hidden.

Fig. 17c-d demonstrate typical problems encountered with gRMSD clustering (R_{max} ranging from 0.5-1.4 Å). It was not possible to choose a set of parameters that would reproduce the proper clustering captured by JP. Fig. 16 demonstrates the issues that are encountered when gRMSD parameters are incremented to find parameters that match JP. A total of 7 systems were examined, all of protein-protein recognition events (see Table 2). In 3 of the systems, there were important interactions that were improperly clustered by gRMSD. In 4 systems, results were the same between JP and gRMSD. In some way, these "easy" cases are appropriate because solvent mapping has been successfully used in SBDD.[5,6,10-14,55] However, to get the correct clustering, each system required individual parameterization. It was necessary to vary the parameters between 0.75-1.25 Å for each individual system. This emphasizes the system-dependant nature of gRMSD clustering; it can often work in a particular case, but parameters are not generalizable. For the other 3 cases, it was not possible to correctly cluster the probes with gRMSD, regardless of our attempts at parameter variation.

For an even presentation, we should note that there was one case where JP failed, an interaction with a glutamic acid side chain in TRAF6-RANK. The cluster is a rare case where the probes are slightly too diffuse to be captured by our JP parameters. However, we also note that gRMSD clustering failed in this case. We were not able to find cases where gRMSD clustering performed better than JP clustering.

Table 2. Comparison of JP and gRMSD clustering across seven protein recognition systems. For each system, we recorded the number of hydrogen-bonding interactions found in the natural binding partner that were near clusters of probes (column 2). Column 3 reports how many of those clusters were identified with JP clustering, column 4 the number found with gRMSD clustering, and column 5 gives the optimal gRMSD cutoff required to identify the most clusters appropriate for each system. The cutoff was increased from 0.25 Å to 1.5 Å in steps of 0.25 Å in order to determine this optimal value. When the cutoff is too generous, the reported clusters are too large and diffuse. We have reported the value that was as large as possible without generating this effect.

Protein System (PDB ID)	# Interactions identified by probes	Properly clustered by JP	Properly clustered by gRMSD	gRMSD cutoff for this system
Barnase-Barstar (1B27) [47]	6	6	4	0.75 Å
CDK6- p16^{INK4a} (1BI7) [50]	5	5	3	0.75 Å
hMms2-hUbc13 (1J7D) [48]	2	2	2	1.25 Å
TRAF6-RANK (1LB5) [49]	2	1	1	1.0 Å
thrombin-BPTI (1BTH) [46]	3	3	3	0.75 Å
CheY-CheA (1EAY) [45]	5	5	5	1.0 Å
HDM2-p53 (1YCR) [44]	6	6	5	0.75 Å

Applications to structure-based drug design

Our group has a history of success modeling HIV-1 protease.[22,23,56] If we re-examine those studies using JP clustering, it generates models that are very comparable to those created by assessing clusters by hand. The structure of the receptor-based pharmacophore models and their performance in identifying known inhibitors over chemically similar decoys are nearly identical (data not shown, but the original models can be found in earlier publications[22]). However, the process is significantly faster using

JP clustering. While manually assessing the models can take *weeks*, the JP clustering takes *15 minutes* on a modest 1.0 GHz Intel Pentium II laptop with 256M of memory. Furthermore, it removes the subjective nature of creating models by hand.

To emphasize the improvement that JP offers, we show in Fig. 18 a case that would not be possible to assess manually. The multiple protein structure (MPS) method[13,23] for SBDD consists of flooding, minimizing, and clustering probes on the surfaces of an ensemble of protein structures. Clusters that are common across the ensemble are identified, and these consensus clusters are used to generate receptor-based pharmacophore models that can be used for lead-generation and SBDD. In developing this model, practitioners of the MPS method must assess clusters for each individual snapshot, choose a representative element from each cluster, and then cluster the representative elements. In Fig 18a, we see the benzene probes from eleven snapshots of an MD simulation[23] (500 probes per snapshot). In Fig 18b, we see the results of applying JP clustering to all 5,500 probes at once (an impossible task by hand): JP clustering results in a model that contains the clusters of the same approximate size and location as the original MPS work. We had to modify the JP parameters in order to cluster such a large system, both increasing the size of the neighbor lists and decreasing the RMSD cutoff. Parameters of (J=250, K_{min}=3, R_{max}=0.9 Å) were used for ethane and benzene, and (J=15, K_{min}=3, R_{max}=0.75 Å) were used for methanol. In line with the original MPS work, we also required clusters to contain at least one element from the beginning, middle, and end of the simulation. This all-in-one technique has not been broadly applied, and the choice of parameters may not be transferable to other systems.

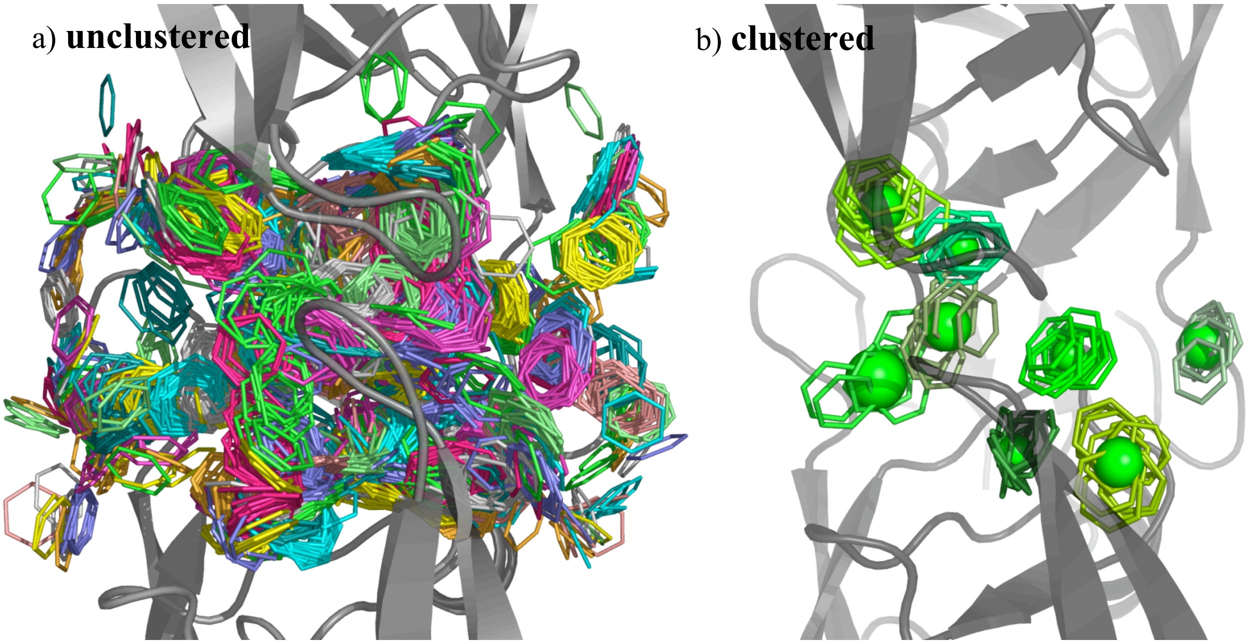


Figure 18. Previously impossible, all-in-one clustering. 5,500 benzene probes are overlaid in part a), 500 each from 11 snapshots of an MD trajectory.[23] All probes from a given snapshot are shown in sticks of the same color. Hydrogens are hidden for clarity, and the protein HIV-1 protease is shown in cartoon. Part b) shows the results of clustering all 5,500 probes at once. Different clusters are shown in different shades of green, and spheres represent the center and RMSD of each cluster.

A final note on the parameters

It is worth recalling the original work of Allen *et al.* in applying JP clustering to the separation of molecules in physical space.[19,20] They found great success with JP clustering and provided a preliminary set of clustering criteria for classifying conformational states of six-membered ring compounds. They noted that their parameters were relevant for the particular systems that they had studied, but that further refinement might be necessary as the technique was applied to more diverse systems. Similarly, our parameters have been successful for clustering the particular systems that we have studied, and we feel that they will provide a general starting point for any similar system. Researchers investigating other systems may find it necessary to modify them slightly, but we do not expect great changes from these parameters. Our automated technique has the great advantage that it is orders of magnitude faster than by-hand clustering and less subjective. Thus, when examining a new system, it is a relatively fast and easy task to verify that a correct set of clustering parameters has been chosen.

Conclusions

We have developed a fast, easy-to-use set of automated procedures to aid in the mapping of protein surfaces. In particular, we have developed improved procedures for placing small-molecule probes near the site of interest. Once those probes have been minimized to the protein surface, our new techniques for clustering the probes shows great success. We have investigated both RMSD-based and JP-based clustering methods, taking the symmetry of the probe molecules into account in all cases. JP-based clustering is significantly faster than previous methods and more accurate.

For validation purposes, we have applied these methods to protein-protein interfaces. We also extended it to our MPS method for SBDD. Our automated techniques produce pharmacophore models that are qualitatively and quantitatively similar to our previous "by-hand" results. These previous results required significant training and time to produce, often taking several weeks for a particular system. The automated methods can perform the same tasks in minutes or hours. Automated methods are also able to cluster thousands of molecules simultaneously, a task that would be impossible by hand.

These methods are robust and may easily be extended to new classes of small molecules. Our automated toolset has been developed as a series of Python scripts and PyMOL plugins and wizards. This makes them easy for other groups to use and extend. Our technique for symmetry-corrected JP clustering could easily be incorporated into other clustering packages such as MCSS, MCSS2PTS, LUDI, or BOSS.[5-7,26,57] The need for better clustering algorithms has been noted in the literature.[14,15]

Acknowledgments

I thank Dr. Kristin L. Meagher for her help throughout this project. I thank Dr. Kelly L. Damm and Dr. Anna L. Bowman for their help in determining the appropriate JP parameters, and I am grateful to Mr. Allen Bailey for maintaining the Linux clusters used extensively in this work. This work was supported by the NIH Molecular Biophysics Training Grant (administered by the University of Michigan), a Beckman Young Investigator Award, and National Institutes of Health grant GM65372.

Supporting information. Details on the importance of symmetry in the calculation of inter-probe distances are given in Appendix 1.

References

(1) Vajda, S.; Guarnieri, F. *Curr. Opin. Drug Di. De.* **2006**, *9*, 354.

(2) Carr, R. A. E.; Congreve, M.; Murray, C. W.; Rees, D. C. *Drug Discov. Today* **2005**, *10*, 987-992.

(3) Erlanson, D. A.; McDowell, R. S.; O'Brien, T. *J. Med. Chem.* **2004**, *47*, 3463-3482.

(4) Rees, D. C.; Congreve, M.; Murray, C. W.; Carr, R. *Nat. Rev. Drug. Discov.* **2004**, *3*, 660-672.

(5) Evensen, E.; Joseph-McCarthy, D.; Karplus, M. **1997**, Harvard University, Cambridge, MA, USA, MCSS version 2.1.

(6) Joseph-McCarthy, D.; Alvarez, J. C. *Proteins Struct. Func. Bioinformat.* **2003**, *51*, 189-202.

(7) Schechner, M.; Dejaegere, A. P.; Stote, R. H. *Int. J. Quantum. Chem.* **2004**, *98*, 378-387.

(8) Hartigan, J. A.; Wong, M. A. *Appl. Stat.-J. Roy. St. C* **1979**, *28*, 100-108.

(9) Jarvis, R. A.; Patrick, E. A. *IEEE T. Comput.* **1973**, *22*, 1025-1034.

(10) Laurie, A. T. R.; Jackson, R. M. *Bioinformatics* **2005**, *21*, 1908-1916.

(11) Miranker, A.; Karplus, M. *Proteins. Struct. Funct. Genet.* **1991**, *11*, 29-34.

(12) Caflisch, A.; Miranker, A.; Karplus, M. *J. Med. Chem.* **1993**, *36*, 2142-2167.

(13) Carlson, H. A.; Masukawa, K. M.; Rubins, K.; Bushman, F. D.; Jorgensen, W. L.; Lins, R. D.; Briggs, J. M.; McCammon, J. A. *J. Med. Chem* **2000**, *43*, 2100-2114.

(14) Schechner, M.; Sirockin, F.; Stote, R. H.; Dejaegere, A. P. *J. Med. Chem.* **2004**, *47*, 4373-4390.

(15) Sirockin, F.; Sich, C.; Improta, S.; Schaefer, M.; Saudek, V.; Froloff, N.; Karplus, M.; Dejaegere, A. *J. Am. Chem. Soc.* **2002**, *124*, 11073-11084.

(16) Arnold, J. R.; Burdick, K. W.; Pegg, S. C.; Toba, S.; Lamb, M. L.; Kuntz, I. D. *J. Chem. Inf. Comput. Sci* **2004**, *44*, 2190-2198.

(17) Kauffman, L.; Rousseeuw, P. J. *Finding Groups in Data: An Introduction to Cluster Analysis*; John Wiley & Sons, Inc.: New York, 1990.

(18) Willett, P. *Similarity and Clustering in Chemical Information Systems*; John Wiley & Sons Inc.: New York, 1987.

(19) Allen, F. H.; Doyle, M. J.; Taylor, R. *Acta Crystallogr. Sect. B Struct. Sci.* **1991**, *47*, 29-40.

(20) Allen, F. H.; Doyle, M. J.; Taylor, R. *Acta Crystallogr. Sect. B Struct. Sci.* **1991**, *47*, 41-49.

(21) Allen, F. H.; Kennard, O.; Taylor, R. *Accounts Chem. Res.* **1983**, *16*, 146-153.

(22) Meagher, K. L.; Lerner, M. G.; Carlson, H. A. *J. Med. Chem* **2006**, *49*, 3478-3484.

(23) Meagher, K. L.; Carlson, H. A. *J. Am. Chem. Soc* **2004**, *126*, 13276-13281.

(24) Lerner, M. G.; Bowman, A. L.; Carlson, H. A. *J. Chem. Inf. Model.* **2007**, *in press*.

(25) Bowman, A. L.; Lerner, M. G.; Carlson, H. A. *J. Am. Chem. Soc.* **2007**, *129*, 3634-3640.

(26) Jorgensen, W. L. **2000**, Yale University, New Haven, CT, BOSS Version 4.2

(27) Jones, S.; Thornton, J. M. *Proc. Natl. Acad. Sci. USA* **1996**, *93*, 13-20.

(28) Pagliaro, L.; Felding, J.; Audouze, K.; Nielsen, S. J.; Terry, R. B.; Krog-Jensen, C.; Butcher, S. *Curr. Opin. Chem. Biol.* **2004**, *8*, 442-449.

(29) Archakov, A. I.; Govorun, V. M.; Dubanov, A. V.; Ivanov, Y. D.; Veselovsky, A. V.; Lewi, P.; Janssen, P. *Proteomics* **2003**, *3*, 380-391.

(30) Keskin, O.; Tsai, C.-J.; Wolfson, H.; Nussinov, R. *Protein Sci.* **2004**, *13*, 1043-1055.

(31) Ofran, Y.; Rost, B. *J. Mol. Biol.* **2003**, *325*, 377-387.

(32) Salwinski, L.; Miller, C. S.; Smith, A. J.; Pettit, F. K.; Bowie, J. U.; Eisenberg, D. *Nucl. Acids Res.* **2004**, *32*, D449-451.

(33) Xenarios, I.; Salwinski, L.; Duan, X. J.; Higney, P.; Kim, S.-M.; Eisenberg, D. *Nucl. Acids Res.* **2002**, *30*, 303-305.

(34) Larsen, T. A.; Olson, A. J.; Goodsell, D. S. *Structure* **1998**, *6*, 421-7.

(35) Reichmann, D.; Rahat, O.; Albeck, S.; Meged, R.; Dym, O.; Schreiber, G. *Proc. Natl. Acad. Sci. USA* **2005**, *102*, 57-62.

(36) DeLano, W. L. *Curr. Opin. Struct. Biol.* **2002**, *12*, 14-20.

(37) Janin, J. *Protein Sci.* **2005**, *14*, 278-283.

(38) Parkinson, J. S.; Kofoid, E. C. *Annu. Rev. Genet.* **1992**, *26*, 71-112.

(39) Ehrlich, L. P.; Nilges, M.; Wade, R. C. *Proteins Struct. Funct. Bioinformat.* **2005**, *58*, 126-133.

(40) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. *Nucl. Acids Res.* **2000**, *28*, 235-242.

(41) Lovell, S. C.; Davis, I. W.; III, W. B. A.; Bakker, P. I. W. d.; Word, J. M.; Prisant, M. G.; Richardson, J. S.; Richardson, D. C. *Proteins. Struct. Funct. Genet.* **2003**, *50*, 437-450.

(42) DeLano, W. L. **2002**, Palo Alto, CA, USA, The PyMOL Molecular Graphics System

(43) Case, D. A.; Pearlman, D. A.; Caldwell, J. W.; Cheatham, T. E., III; Ross, W. S.; Simmerling, C. L.; Darden, T. A.; Merz, K. M.; Stanton, R. V.; Cheng, A. L. **1999**, University of California, San Francisco, AMBER 6

(44) Kussie, P. H.; Gorina, S.; Marechal, V.; Elenbaas, B.; Moreau, J.; Levine, A. J.; Pavletich, N. P. *Science* **1996**, *274*, 948-953.

(45) McEvoy, M. M.; Hausrath, A. C.; Randolph, G. B.; Remington, S. J.; Dahlquist, F. W. *Proc. Natl. Acad. Sci. USA* **1998**, *95*, 7333-7338.

(46) Locht, A. v. d.; Bode, W.; Huber, R.; Bonniec, B. F. L.; Stone, S. R.; Esmon, C. T.; Stubbs, M. T. *EMBO J.* **1997**, *16*, 2977-2984.

(47) Buckle, A. M.; Schreiber, G.; Fersht, A. R. *Biochemistry* **1994**, *33*, 8878-8889.

(48) Moraes, T. F.; Edwards, R. A.; McKenna, S.; Pastushok, L.; Xiao, W.; Glover, J. N. M.; Ellison, M. J. *Nat. Struct. Mol. Biol.* **2001**, *8*, 669-673.

(49) Ye, H.; Arron, J. R.; Lamothe, B.; Cirilli, M.; Kobayashi, T.; Shevde, N. K.; Segal, D.; Dzivenu, O. K.; Vologodskaia, M.; Yim, M.; Du, K.; Singh, S.; Pike, J. W.; Darnay, B. G.; Choi, Y.; Wu, H. *Nature* **2002**, *418*, 443-447.

(50) Russo, A. A.; Tong, L.; Lee, J.-O.; Jeffrey, P. D.; Pavletich, N. P. *Nature* **1998**, *395*, 237-243.

(51) Crowley, P. B.; Golovin, A. *Proteins Struct. Funct. Bioinformat.* **2005**, *59*, 231-239.

(52) Jorgensen, W. L.; Tirado-Rives, J. *J. Am. Chem. Soc.* **1988**, *110*, 1657-1666.

(53) Duda, R. O.; Hart, P. E.; Stork, D. G. *Pattern Classification*; Wiley-Interscience, 2000.

(54) Bowman, A. L.; Nikolovska-Coleska, Z.; Zhong, H.; Wang, S.; Carlson, H. A. *J. Am. Chem. Soc.* **2007**, *129*, 12809-12814.

(55) Carlson, H. A.; Masukawa, K. M.; McCammon, J. A. *J. Phys. Chem. A* **1999**, *103*, 10213-10219.

(56) Damm, K. L.; Carlson, H. A. *J. Am. Chem. Soc.* **2007**, *129*, 8225-8235.

(57) Böhm, H. J. *J. Comput-Aided. Mol. Des.* **1992**, *6*, 61-78.

Appendix 1

Supplemental information

Supplemental information from Chapter 2

Coordinates and RMSD for the pharmacophore models (radii of the elements are determined from the RMSD).

Open-loop 1-ns, coordinates provided are relative to 1RA1.

Element Type	x	y	z	RMSD (Å)
Excluded Volume	4.5111	2.5786	12.5531	1.5
Excluded Volume	1.8823	3.5392	5.6876	1.5
Excluded Volume	10.3746	-0.749	0.2989	1.5
Excluded Volume	1.4852	-2.7131	11.6940	1.5
Aro|Hyd	7.703	-1.5424	5.6534	1.1844
Aro	3.0593	0.2795	7.5955	1.1750
Aro|Hyd	2.2915	4.5311	-0.2134	1.0608
Don	0.9677	-0.2109	6.0317	0.7321
Acc	1.9332	5.4698	0.4778	0.5106

Open-loop 2-ns, coordinates provided are relative to 1RA1.

Element Type	x	y	z	RMSD (Å)
Excluded Volume	4.4668	2.5432	12.5663	1.5
Excluded Volume	1.8582	3.5692	5.6852	1.5
Excluded Volume	10.3973	-0.7739	0.1583	1.5
Excluded Volume	1.4767	-2.7000	11.5401	1.5
Aro|Hyd	7.6449	-1.4136	5.6133	1.6122
Aro|Hyd	3.3476	0.2932	7.8253	1.1818
Aro|Hyd	2.2022	4.5291	-0.1608	1.0746
Don	0.9071	-0.2803	5.7418	0.7415
Acc	2.0309	5.4093	0.4662	0.8543

Open-loop 4-ns, coordinates provided relative to 1RA1

Element Type	x	y	z	RMSD (Å)
Excluded Volume	4.3666	2.502	12.6023	1.5
Excluded Volume	1.9072	3.6145	5.7946	1.5
Excluded Volume	10.3774	-0.754	-0.0978	1.5
Excluded Volume	1.5150	-2.8302	11.6647	1.5
Aro|Hyd	2.1599	4.4635	-0.0923	1.0793
Aro|Hyd	3.0418	0.4179	8.2954	0.8730
Aro|Hyd	6.8885	0.2456	5.5951	1.2287
Acc	2.222	5.304	0.6629	0.8435
Don	0.9228	-0.2589	5.7279	0.7106
Don	0.0741	1.2701	8.6283	0.7317
Don	3.7481	-0.3463	8.0185	0.7929

Closed-loop 1-ns, coordinates provided are relative to 1RX1

Element Type	x	y	z	RMSD (Å)
Excluded Volume	4.2689	2.9744	12.3521	1.5
Excluded Volume	1.8389	3.7252	5.5529	1.5
Excluded Volume	10.3577	-0.9057	0.1451	1.5
Excluded Volume	1.6675	-2.5312	11.915	1.5
Aro|Hyd	7.4154	-1.4043	4.8614	1.6313
Aro|Hyd	2.6	4.5286	-1.0953	1.3577
Aro|Hyd	3.4466	0.5377	8.052	1.0810
Don	3.8501	0.1714	8.2825	0.5993
Don	0.93	-0.2216	5.3727	0.6030
Acc	2.3978	4.8414	0.8422	0.3366

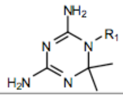
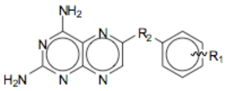
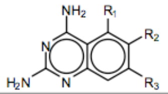
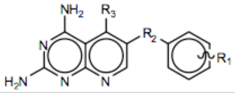
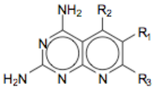
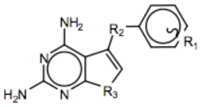
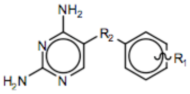
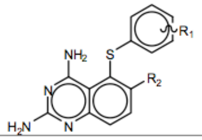
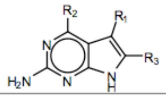
Closed-loop 2-ns, coordinates provided are relative to 1RX1

Element Type	x	y	z	RMSD (Å)
Excluded Volume	4.2366	2.9505	12.2683	1.5
Excluded Volume	1.8048	3.7348	5.6265	1.5
Excluded Volume	10.3898	-0.8004	0.0127	1.5
Excluded Volume	1.5321	-2.5023	11.9865	1.5
Aro|Hyd	2.6953	4.5622	-1.0938	1.3388
Aro	3.7203	0.4451	7.4092	0.6791
Aro|Hyd	6.6335	0.4768	5.3257	0.9976
Acc	2.6019	5.1799	0.1112	0.8899
Don	1.1476	-0.0249	5.8053	0.8837
Don	3.766	0.169	8.3395	0.7854

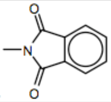
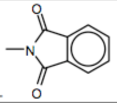
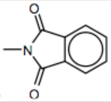
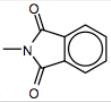
Closed-loop 4-ns, coordinates provided are relative to 1RX1

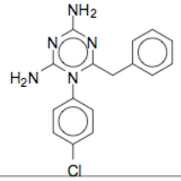
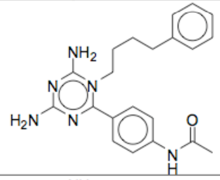
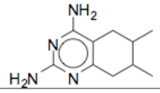
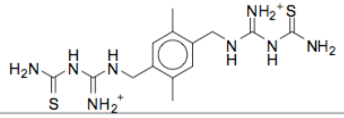
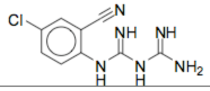
Element Type	x	y	z	RMSD (Å)
Excluded Volume	4.175	2.9441	12.2428	1.5
Excluded Volume	1.7944	3.77	5.6002	1.5
Excluded Volume	10.2912	-0.8796	0.0704	1.5
Excluded Volume	1.5587	-2.5454	12.0047	1.5
Aro	3.6295	0.4727	7.4659	0.6377
Aro	7.1276	-1.819	4.0098	1.4987
Don	3.7982	0.0126	8.3552	0.7367
Acc	2.7017	5.3018	-0.1119	0.8707
Don	1.2495	-0.2748	5.8424	1.2593
Don	-0.247	2.0104	8.7224	0.7831

E. coli DHFR inhibitors

B 	D 	G 	K 
L 	M 	N 	O 
U 			

Name	R_1	R_2	R_3		$IC_{50}EC$	Ref	High Aff?
37	$-C_6H_4$-3-Cl			B	0.6026	Ghose *et al.* (1985)	
38	$-C_6H_3$-3,4-Cl			B	0.1585	Ghose *et al.* (1985)	
39	$-(CH_2)_3$-Ph			B	0.0631	Ghose *et al.* (1985)	
VI	$-(CH_2)_2CHMe_2$			B	0.69	Baker (1967)	
VII	$-(CH_2)_5Me$			B	0.92	Baker (1967)	
VIII	$-(CH_2)_7Me$			B	0.27	Baker (1967)	
XIII	$-(CH_2)_4Ph$			B	0.21	Baker (1967)	
XXX	$-C_6H_4$-3-CH_2-Ph			B	0.28	Baker (1967)	
XXXII	$-C_6H_4$-3-$(CH_2)_2$-Ph			B	0.092	Baker (1967)	
XXXVII	$-C_6H_4$-4-Cl			B	0.40	Baker (1967)	
Methotrexate	-4-CONHCH$(CO_2H)CH_2CH_2CO_2H$	$-CH_2N(Me)-$		D	0.016	Gangjee *et al.* (2005)	EC
3	-O-C_6H_4-4-F	-H	-H	G	0.40	Zolli-Juran *et al.* (2003)	
4	-O-C_6H_4-4-Me	-H	-H	G	0.19	Zolli-Juran *et al.* (2003)	
5	$-OCH_2$-C_6H_4-4-CF_3	-H	-H	G	0.19	Zolli-Juran *et al.* (2003)	
11	-3,4,5-OMe	$-NHCH_2-$	-H	K	0.76	Gangjee *et al.* (1996)	
21	-2,5-OMe	$-NHCH_2-$	-H	K	0.13	Gangjee *et al.* (1996)	
22	-2,5-OMe	$-N(Me)CH_2-$	-H	K	0.14	Gangjee *et al.* (1996)	
25	-H	$-CH_2-$	-H	K	0.40	Hurlbert *et al.* (1968)	
29	-4-CONHCH$(CO_2H)CH_2CH_2CO_2H$	$-NHCH_2-$	-H	K	0.013	Gangjee *et al.* (1996)	EC
30	-4-CONHCH$(CO_2H)CH_2CH_2CO_2H$	$-N(Me)CH_2-$	-H	K	0.013	Gangjee *et al.* (1996)	EC
12	-Et	-H	-Ph	L	0.14	Hurlbert *et al.* (1968)	

15	-CH$_2$CH$_2$Me	-H	-CH$_2$CH$_2$CH$_2$Me	L	0.019	Hurlbert *et al.* (1968)	EC
18	-(CH$_2$)$_3$Me	-H	-H	L	0.50	Hurlbert *et al.* (1968)	
35	-Me	-Me	-H	L	0.23	Hurlbert *et al.* (1968)	
37	-CH$_2$CH$_2$Me	-Me	-H	L	0.015	Hurlbert *et al.* (1968)	EC
38	-(CH$_2$)$_3$Me	-Me	-H	L	0.02	Hurlbert *et al.* (1968)	EC
39	-CH(Me)Et	-Me	-H	L	0.002	Hurlbert *et al.* (1968)	EC
40	-(CH$_2$)$_4$Me	-Me	-H	L	0.38	Hurlbert *et al.* (1968)	
43	-(CH$_2$)$_5$Me	-Me	-H	L	0.0092	Hurlbert *et al.* (1968)	EC
44	-(CH$_2$)$_6$Me	-Me	-H	L	0.0078	Hurlbert *et al.* (1968)	EC
45	-CH(Me)-Ph	-Me	-H	L	0.01	Hurlbert *et al.* (1968)	EC
1	-4-CONHCH(CO$_2$H)CH$_2$CH$_2$CO$_2$H	-CH$_2$CH$_2$-	-O-	M	1.00	Gangjee *et al.* (2000)	
2	-4-CONHCH(CO$_2$H)CH$_2$CH$_2$CO$_2$H	-CH$_2$NH-	-O-	M	0.45	Gangjee *et al.* (2000)	
3	-4-CONHCH(CO$_2$H)CH$_2$CH$_2$CO$_2$H	-CH$_2$N(Me)-	-O-	M	0.22	Gangjee *et al.* (2000)	
5	-4-CONHCH(CO$_2$H)CH$_2$CH$_2$CO$_2$H	-CH$_2$CH(Me)-	-O-	M	0.42	Gangjee *et al.* (2000)	
2	-H	-CH$_2$-		N	0.6607	Ghose *et al.* (1985)	EC
3	-4-NO$_2$	-CH$_2$-		N	0.6310	Ghose *et al.* (1985)	EC
4	-3-CH$_2$OH	-CH$_2$-		N	0.5248	Ghose *et al.* (1985)	
5	-4-NH$_2$	-CH$_2$-		N	0.5012	Ghose *et al.* (1985)	
6	-4-Cl	-CH$_2$-		N	0.3548	Ghose *et al.* (1985)	
7	-3,4-OH	-CH$_2$-		N	0.3467	Ghose *et al.* (1985)	
8	-3-OH	-CH$_2$-		N	0.3388	Ghose *et al.* (1985)	
9	-4-Me	-CH$_2$-		N	0.3311	Ghose *et al.* (1985)	
10	-4-OCF$_3$	-CH$_2$-		N	0.2692	Ghose *et al.* (1985)	
11	-3-CH$_2$Me	-CH$_2$-		N	0.2570	Ghose *et al.* (1985)	
12	-3-Cl	-CH$_2$-		N	0.2239	Ghose *et al.* (1985)	
13	-3-Me	-CH$_2$-		N	0.1995	Ghose *et al.* (1985)	
14	-4-N(Me)$_2$	-CH$_2$-		N	0.1660	Ghose *et al.* (1985)	
15	-4-OMe	-CH$_2$-		N	0.1514	Ghose *et al.* (1985)	
16	-4-Br	-CH$_2$-		N	0.1514	Ghose *et al.* (1985)	
17	-4-NHCOMe	-CH$_2$-		N	0.1288	Ghose *et al.* (1985)	
18	-4-OSO$_2$Me	-CH$_2$-		N	0.1202	Ghose *et al.* (1985)	
19	-3-OMe	-CH$_2$-		N	0.1175	Ghose *et al.* (1985)	
20	-3-Br	-CH$_2$-		N	0.1096	Ghose *et al.* (1985)	
21	-3-CF$_3$	-CH$_2$-		N	0.0955	Ghose *et al.* (1985)	
22	-3-CF$_3$ -4-OMe	-CH$_2$-		N	0.0204	Ghose *et al.* (1985)	EC
23	-3,4-OMe	-CH$_2$-		N	0.0191	Ghose *et al.* (1985)	EC
24	-3,5-OMe	-CH$_2$-		N	0.0042	Ghose *et al.* (1985)	EC
2	-3,5-OMe,-4-OH	-CH$_2$-		N	0.0110	Roth *et al.* (1981)	
3	-3,5-OMe,-4-OEt	-CH$_2$-		N	0.0075	Roth *et al.* (1981)	
4	-3,5-OMe,-4-OCH$_2$CH$_2$Me	-CH$_2$-		N	0.013	Roth *et al.* (1981)	EC
5	-3,5-OMe,-4-OCH(Me)$_2$	-CH$_2$-		N	0.024	Roth *et al.* (1981)	EC
6	-3,5-OMe,-4-OCH$_2$CH=CH$_2$	-CH$_2$-		N	0.0052	Roth *et al.* (1981)	EC
7	-3,5-OMe,-4-OC(Me)$_3$	-CH$_2$-		N	0.02	Roth *et al.* (1981)	EC
8	-3,5-OMe,-4-OCH(Me)Et	-CH$_2$-		N	0.016	Roth *et al.* (1981)	EC
9	-3,5-OMe,-4-O(CH$_2$)$_4$Me	-CH$_2$-		N	0.015	Roth *et al.* (1981)	EC
10	-3,5-OMe,-4-O(CH$_2$)$_5$Me	-CH$_2$-		N	0.0042	Roth *et al.* (1981)	EC
11	-3,5-OMe,-4-O(CH$_2$)$_7$Me	-CH$_2$-		N	0.006	Roth *et al.* (1981)	EC
12	-3,5-OMe,-4-O(CH$_2$)$_2$OH	-CH$_2$-		N	0.022	Roth *et al.* (1981)	EC
13	-3,5-OMe,-4-O(CH$_2$)$_2$OMe	-CH$_2$-		N	0.018	Roth *et al.* (1981)	EC
14	-3,5-OMe,-4-O(CH$_2$)$_2$Cl	-CH$_2$-		N	0.015	Roth *et al.* (1981)	EC
15	 -3,5-OMe,-4-OCH$_2$CH$_2$-	-CH$_2$-		N	0.018	Roth *et al.* (1981)	EC
16	-3,5-OMe,-4-O(CH$_2$)$_2$NH$_2$	-CH$_2$-		N	0.022	Roth *et al.* (1981)	EC
17	-3,5-OMe,-4-O(CH$_2$)$_2$-morpholino	-CH$_2$-		N	0.014	Roth *et al.* (1981)	EC
18	-3,5-OMe,-4-OCH$_2$COEt	-CH$_2$-		N	0.017	Roth *et al.* (1981)	EC
19	-3,5-OMe,-4-OCH$_2$CO$_2$H	-CH$_2$-		N	0.24	Roth *et al.* (1981)	
20	-3,5-OMe,-4-O(CH$_2$)$_3$OH	-CH$_2$-		N	0.0042	Roth *et al.* (1981)	EC
21	-3,5-OMe,-4-O(CH$_2$)$_3$Cl	-CH$_2$-		N	0.005	Roth *et al.* (1981)	EC
22	 -3,5-OMe,-4-O(CH$_2$)$_3$-	-CH$_2$-		N	0.018	Roth *et al.* (1981)	EC
23	-3,5-OMe,-4-O(CH$_2$)$_3$NH$_2$	-CH$_2$-		N	0.046	Roth *et al.* (1981)	
24	-3,5-OMe,-4-O(CH$_2$)$_3$-1-piperidine	-CH$_2$-		N	0.046	Roth *et al.* (1981)	
25	-3,5-OMe,-4-O(CH$_2$)$_3$-1-piperazine-4-Me	-CH$_2$-		N	0.005	Roth *et al.* (1981)	EC
26	-3,5-OMe,-4-O-C$_6$H$_4$-4-NO$_2$	-CH$_2$-		N	0.0078	Roth *et al.* (1981)	EC
27	-3,5-OMe,-4-O-C$_6$H$_4$-4-NH$_2$	-CH$_2$-		N	0.0063	Roth *et al.* (1981)	EC
28	-3,5-OMe,-4-O-C$_6$H$_4$-4-NHCOCH$_2$Br	-CH$_2$-		N	0.0055	Roth *et al.* (1981)	EC
29	-3,5-OMe,-4-O-CH$_2$C$_6$H$_4$	-CH$_2$-		N	0.014	Roth *et al.* (1981)	EC
30	-3,5-OMe,-4-O-C$_6$H$_4$-4-NO$_2$	-CH$_2$-		N	0.008	Roth *et al.* (1981)	EC
31	-3,5-OMe,-4-O-C$_6$H$_4$-3,4,5-OMe	-CH$_2$-		N	0.02	Roth *et al.* (1981)	EC
32	 -3,5-OMe,-4-O(CH$_2$)$_4$-	-CH$_2$-		N	0.028	Roth *et al.* (1981)	EC
33	 -3,5-OMe,-4-O(CH$_2$)$_6$-	-CH$_2$-		N	0.04	Roth *et al.* (1981)	
34	-3,5-OMe,-4-O(CH$_2$)$_6$NH$_2$	-CH$_2$-		N	0.015	Roth *et al.* (1981)	EC
35	-3,5-OMe,-4-O(CH$_2$)$_5$CO$_2$Me	-CH$_2$-		N	0.062	Roth *et al.* (1981)	
36	-3,5-OMe,-4-O(CH$_2$)$_5$CO$_2$H	-CH$_2$-		N	0.02	Roth *et al.* (1981)	EC

37	-3,5-OMe,-4-OCH$_2$CHOHCH$_2$OH	-CH$_2$-		N	0.014	Roth *et al.* (1981)	EC
37	-3,5-OMe,-4-OCH$_2$CHOHCH$_2$OH	-CH$_2$-		N	0.014	Roth *et al.* (1981)	EC
38	-3,5-OMe,-4-OCH$_2$-oxirane	-CH$_2$-		N	0.014	Roth *et al.* (1981)	EC
46a	-3,5-OMe,-4-Me	-CH$_2$-		N	0.0074	Roth *et al.* (1981)	EC
46b	-3,5-OMe,-4-Et	-CH$_2$-		N	0.0099	Roth *et al.* (1981)	EC
48	-3,5-OMe,-4-O(CH$_2$)$_3$Me	-CH$_2$-		N	0.01	Roth *et al.* (1981)	EC
Trimethoprim	-3,4,5-OMe	-CH$_2$-		N	0.018	Zolli-Juran *et al.* (2003)	EC
1	-4-Me	-H		O	0.31	Zolli-Juran *et al.* (2003)	
2	-4-F	-H		O	0.32	Zolli-Juran *et al.* (2003)	
3	-Me	-NH$_2$	-S-C$_6$H$_4$-4-CONHCH(CO$_2$H)(CH$_2$)$_2$CO2H	U	0.0066	Gangjee *et al.* (2005)	EC
XLII					0.62	Baker (1967)	
XLIV					0.41	Baker (1967)	
7					0.79	Zolli-Juran *et al.* (2003)	
10	 NH2+ S N H N H NH2 H2N H N H N S NH2+				0.109	Zolli-Juran *et al.* (2003)	
12					0.32	Zolli-Juran *et al.* (2003)	

Baker, BR (1967). Differential Binding to Hydrophobic Bonding Region of T2 Phage Induced Escherichia Coli B and Pigeon Liver Dihydrofolic Reductases. J. Med. Chem. **10**(5) 912-917.

Gangjee, A, A Vasudevan, SF Queener and RL Kisliuk (1996). 2,4-Diamino-5-deaza-6-substituted pyrido 2,3-d pyrimidine antifolates as potent and selective nonclassical inhibitors of dihydrofolate reductases. J. Med. Chem. **39**(7) 1438-1446.

Gangjee, A, YB Zeng, JJ McGuire and RL Kisliuk (2000). Effect of C9-methyl substitution and C8-C9 conformational restriction on antifolate and antitumor activity of classical 5-substituted 2,4-diaminofuro[2,3-d] pyrimidines. J. Med. Chem. **43**(16) 3125-3133.

Gangjee, A, X Lin, RL Kisliuk and JJ McGuire (2005). Synthesis of N-{4-[(2,4-diamino-5-methyl-4,7-dihydro-3H-pyrrolo[2,3-d]pyrimidin-6-yl) thio]benzoyl}-L-glutamic acid and N-{4-[(2-amino-4-oxo-5-methyl-4,7-dihydro-3H-pyrrolo[2,3-d]pyrimidin-6-y l)thio]benzoyl}-L-glutamic acid as dual inhibitors of dihydrofolate reductase and thymidylate synthase and as potential antitumor agents. J. Med. Chem. **48**(23) 7215-7222.

Ghose, AK and GM Crippen (1985). Use of Physicochemical Parameters in Distance Geometry and Related Three-Dimensional Quantitative Structure-Activity Relationships: A Demonstration Using Escherichia coli Dihydrofolate Reductase Inhibitors. J. Med. Chem. **28**(3) 333-346.

Hurlbert, BS, R Ferone, TA Herrmann, Hitching.Gh, M Barnett and SRM Bushby (1968). Studies on Condensed Pyrimidine Systems .25. 2,4-Diaminopyrido[2,3-D]Pyrimidines . Biological Data. J. Med. Chem. **11**(4) 711-717.

Roth, B, E Aig, BS Rauckman, JZ Strelitz, AP Phillips, R Ferone, SRM Bushby and CW Sigel (1981). 2,4-Diamino-5-Benzylpyrimidines and Analogs as Anti-Bacterial Agents .5. 3',5'-Dimethoxy-4'-Substituted-Benzyl Analogs of Trimethoprim. J. Med. Chem. **24**(8) 933-941.

Zolli-Juran, M, JD Cechetto, R Hartlen, DM Daigle and ED Brown (2003). High throughput screening identifies novel inhibitors of Escherichia coli dihydrofolate reductase that are competitive with dihydrofolate. Bioorg. Med. Chem. Lett. **13**(15) 2493-2496.

Supplemental information from Chapter 3

Correlated dynamics, the loops, and the network

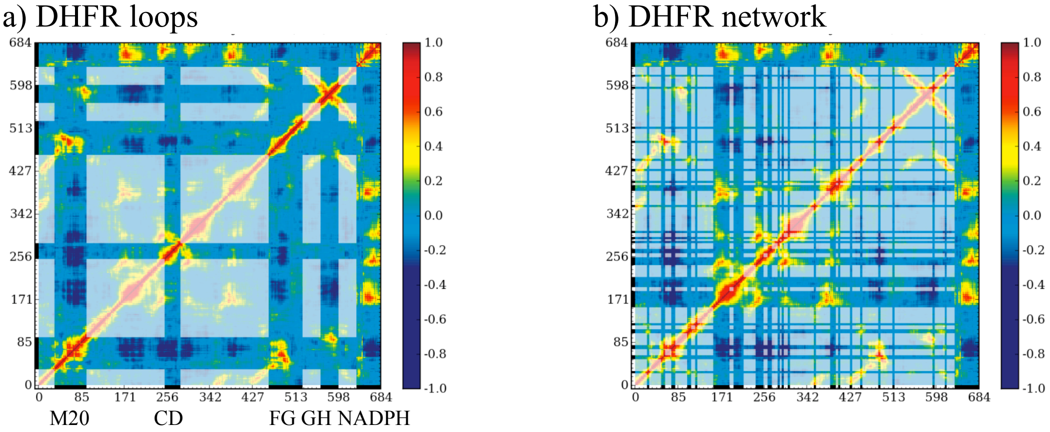


Figure A1: Highlighting loops and networks. Part (a) highlights the m20, CD, FH, and GH loops, as well as the ligand. We can see that almost the entirety of the anti-correlated motion is localized to these regions. Part (b) highlights all residues that have been identified as participating in networks of correlated motions in *E. coli* DHFR. We can see that all of these residues are involved in either correlated or anti-correlated motions, with most of them involved in anti-correlated motions. The network is comprised of residues 7, 14, 15, 19, 20, 27, 28, 31, 40, 41, 42, 43, 44, 45, 46, 47, 50, 51, 52, 53, 54, 61, 62, 63, 64, 67, 68, 72, 74, 77, 90, 98, 99, 100, 102, 108, 113, 122, 129, 149, 155, 160.[1-9]

Hydrogen bonds and bridging water

The default ptraj parameters of 120-degree angle cutoff and a distance cutoff of 3.5Å were used to define hydrogen bonds between appropriate donor and acceptor atoms. The same definitions of hydrogen bonds were used for bridging water molecules, and a water molecule was defined as bridging if it formed hydrogen bonds to two or more protein or cofactor atoms at the same time.

Comparisons to the traditional network known for DHFR

Hydrogen bonds are known to contribute to the reaction coordinate, the stabilization of the M20 loop conformations, and correlated dynamics. The Asp122 N-H → Gly15 O hydrogen bond is quite significant, as the motion of these two residues relative to each other contributes to the reaction coordinate.[3,10] We observe this bond during most (65%) of both the open-loop and closed-loop simulations. In addition to the hydrogen bond, we find significant correlated motion between the two residues. They are found in an area of correlated, rather than anti-correlated motion, which is accentuated during the activated dynamics, and both residues show significant anti-correlations to NADPH. Similarly, Thorpe *et al.* have shown that the hydrogen bond between Ala19 and the NADPH O'N2 correlates with the reaction coordinate.[10] We see this interaction approximately 5% of the time in both of our simulations, while it appears for 28% of their wild-type simulation and 10-17% of their mutant simulations. Interestingly, we find Ala19 hydrogen-bonding to the nearby NADPH O'N3 18% of the time in both of our simulations. The closed conformation is known to be stabilized by hydrogen bonds from the amide backbones of Gly15 and Glu17 in the M20 loop to Asp122 in the FG loop.[3] We see the see Glu17 N-H

→ Asp122 OD1/2 hydrogen bond less frequently (33% of the closed-loop simulation and only very briefly during the open-loop simulation) than the Asp122 N-H → Gly15 O hydrogen bond, but we find that Asp122, Gly15, and Glu17 are all involved in significant positively-correlated motions with each other. It is interesting to note that in the closed-loop simulation, when Glu17 N-H is not hydrogen bonding to Asp122, it is very often hydrogen bonding to NADPH O'N3. That hydrogen bond is never seen in the open-loop simulation. The presence of these hydrogen bonds gives further evidence that our DHFR•NADPH simulations produce motions that can be coordinated with progress along the reaction coordinate.

In support of the new site

In the closed-loop simulation, a water bridge between the backbone oxygen of Glu154 and the side chain of Trp30 persists for 46% of the simulation with an average lifetime of 157 ps. Trp30 is alternatively seen in a second bridging-water interaction with the side chain of Thr113 (18% occupancy and an average lifetime of 34 ps).

Supplemental information from Chapter 5

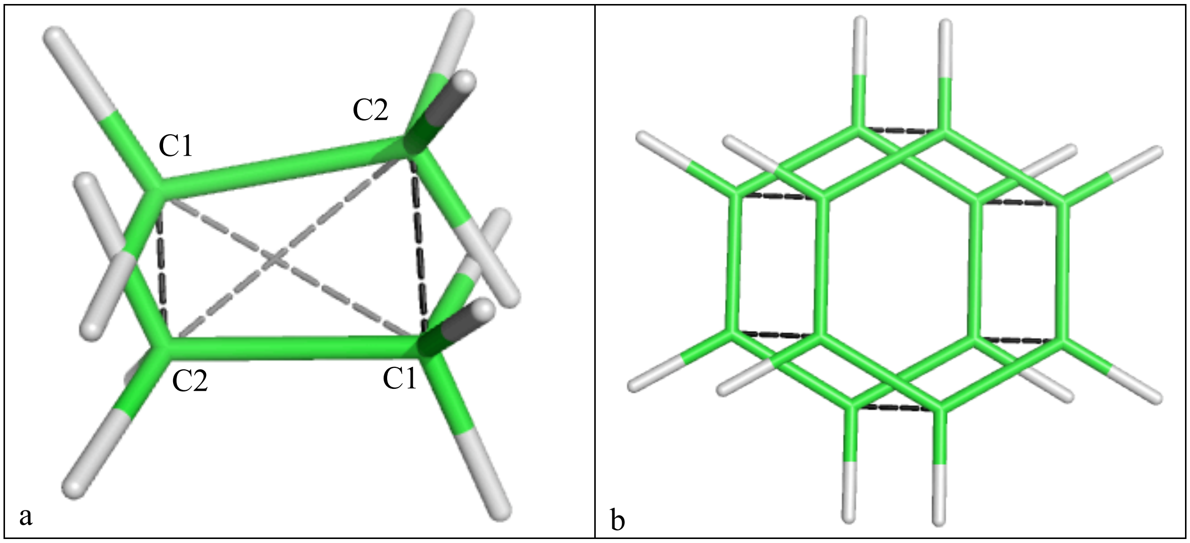


Figure 2: Calculating the distance between probe molecules. a) The carbon atoms in an ethane molecule are indistinguishable but are labeled C1 and C2. We must examine all renamings of the carbons so that we calculate the correct (black) distances rather than the incorrect (grey) ones. b) Benzene exhibits a similar problem. The six carbons can be relabeled in 12 unique ways via symmetrically equivalent rotations and mirroring. We account for this in our symmetry-corrected RMSD calculations.

References

(1) Agarwal, P. K.; Billeter, S. R.; Hammes-Schiffer, S. *J. Phys. Chem. B* **2002**, *106*, 3283-3293.

(2) Pan, H.; Lee, J. C.; Hilser, V. J. *Proc. Natl. Acad. Sci. USA* **2000**, *97*, 12020-12025.

(3) Agarwal, P. K.; Billeter, S. R.; Rajagopalan, P. T. R.; Benkovic, S. J.; Hammes-Schiffer, S. *Proc. Natl. Acad. Sci. USA* **2002**, *99*, 2794-2799.

(4) Benkovic, S. J.; Hammes-Schiffer, S. *Science* **2003**, *301*, 1196-1202.

(5) Wang, L.; Tharp, S.; Selzer, T.; Benkovic, S. J.; Kohen, A. *Biochemistry* **2006**, *45*, 1383-1392.

(6) Watney, J. B.; Hammes-Schiffer, S. *J. Phys. Chem. B* **2006**, *110*, 10130-10138.

(7) Wong, K. F.; Selzer, T.; Benkovic, S. J.; Hammes-Schiffer, S. *Proc. Natl. Acad. Sci. USA* **2005**, *102*, 6807-6812.

(8) Wong, K. F.; Watney, J. B.; Hammes-Schiffer, S. *J. Phys. Chem. B* **2004**, *108*, 12231-12241.

(9) Hammes-Schiffer, S.; Benkovic, S. J. *Annu. Rev. Biochem.* **2006**, *75*, 519-541.

(10) Thorpe, I. F.; Brooks, C. L., III *Proteins: Struct. Funct. Bioinformatics* **2004**, *57*, 444-457.

Appendix 2

Source code

Directory listing 1

The PyPAT project is organized as a series of Python scripts. In order for module imports to work correctly, it is necessary to preserve the directory structure.

```
phar20-217~/work/PyPAT/code$ ls -R
Readme.txt
pypat
drivers

./drivers:
collect_water_bridges.py
convert_to_numpy_format.py
display_bridging_interactions.py
do_correlated_md_analysis.py
make_correlated_dynamics_plots.py
make_movies.py
parse_sander_output.py
run_ptraj.py
write_ptraj_input_files.py

./pypat:
__init__.py
hbond
md_analysis_utils.py
plotting.py
sentinel_map.py
tool_utils.py

./pypat/hbond:
__init__.py
hbond_analysis.py
hbond_definitions.py
pymol_hbond_analysis.py
```

Files 1

Readme.txt

```
Installation
------------

The executable scripts are located in the "drivers" subdirectory, and
the libraries are located in the "pypat" directory. Inside each of the
scripts, you will need to edit the lines containing
"PYPAT_CODE_DIRECTORY" so that they point to the directory in which
the PyPAT package is installed.

Dependencies
------------

Python 2.5 or greater is required.
```

```
Your Python installation must have the following installed:

 - matplotlib 0.9.0
 - numpy 1.0.1
 - PyMOL 1.0

And you must have following installed on your system:

 - gnuplot
 - ImageMagick 6.3.2
 - PyMOL 1.0

Running the Tools
-----------------

The executable scripts are found in the "drivers" subdirectory. You
may wish to place this directory in your PATH. All scripts support the
--help command line option. Documentation for individual tools
follows.
```

drivers/collect_water_bridges.py

```
#!/usr/bin/env python

import sys
if sys.platform == 'darwin':
    PYPAT_CODE_DIR = '/Users/mglerner/work/PyPAT/code/'
elif sys.platform == 'linux2':
    PYPAT_CODE_DIR = '/users/mlerner/work/src/PyPAT/code/'
else:
    sys.exit('Unknown system type')

sys.path.append(PYPAT_CODE_DIR)


if __name__ == '__main__':
    from pypat.hbond import pymol_hbond_analysis
    from optparse import OptionParser

    usage = """
    Run this like:

    pymol -qcr collect_water_bridges.py -- --name=myprotein --dist-cutoff=3.5

    Do not forget the double dashes after the script name.
    """
    parser = OptionParser(usage=usage)

    parser.add_option("--name",'-n',dest="name",default="nrna",
                      help="Trajectory and topology must be named name.trj and name.top
respectively. [default: %default]"
                      )
    parser.add_option("--chunk-size",'-c',dest="chunksize",default=5,#default=500,
                      help="How many MD steps to process at a time. If you do too many at
a time, PyMOL will slow down. Too few, and you're wasting time starting/stopping PyMOL.
[default: %default%]",
                      type='int',
                      )
    parser.add_option("--num-steps",'-s',dest="numsteps",default=10,#default=500,
                      help="number of steps in your MD trajectory. [default: %default]",
                      type='int',
                      ##FIXME: can be extracted from PyMOL
                      )
    parser.add_option("--dist-cutoff",'-d',dest="distcutoff",default=4.0,
                      help="Heavy atom to heavy atom distance cutoff. [default:
%default]",
                      type='float',
                      )
    parser.add_option('--angle-cutoff','-a',dest='anglecutoff',default=0.0,
                      help="Angle cutoff. If heavy:hydro:heavy angle must be greater than
this. [default: %default]",
                      type='float',
                      )
    #argv = sys.argv[sys.argv.index('--') + 1:]
    try:
        argv = sys.argv
    except AttributeError:
        argv = pymol_argv
        sys.argv = pymol_argv # this is necessary for optparse to handle the --help
option.
    try:
        argv = argv[argv.index('--') + 1:]
    except IndexError:
        argv = []

    options,args = parser.parse_args(args=argv)


    r = range(1,options.numsteps+1,options.chunksize)
    if r[-1] != options.numsteps:
        r.append(options.numsteps)
    starts_and_stops = zip(r[:-1],r[1:])
    print starts_and_stops
    for (start,stop) in starts_and_stops:
        print
        print 'DOING',start,stop
        print
        pymol_hbond_analysis.find_bridging_waters_in_trajectory(name=options.name,
                                                                start=start,
                                                                stop=stop,

hbond_dist_cutoff=options.distcutoff,

hbond_angle_cutoff=options.anglecutoff,
                                                                )
```

drivers/convert_to_numpy_format.py

```
#!/usr/bin/env python

import glob,bz2
import scipy
import scipy.io
import os


if __name__ == '__main__':
    from optparse import OptionParser
    usage = """This will convert the .dat or .dat.bz2 files to numpy versions.
    It will not automatically delete the .dat(.bz2) files.  If your input
    files are bz2, your output files will be too.

    If you already have a corresponding .numpy or .numpy.bz2 file, we won't
    write out a new file.
    """
    parser = OptionParser(usage=usage)

    parser.add_option('--dir',dest='dir',default='.',
                      help="Directory in which the files reside. [default: %default]")
    parser.add_option("--structure-name",dest="structurename",
                      default="1rx1",
                      help="Name of your structure.  E.g. 1RX1 or 1SGZ")
    parser.add_option('--compression',dest='compression',
                      default='',
                      help="Type of compression currently used on files.  Leave blank for
uncompressed, .bz2 if they're .bz2 files. Note that it's '.bz2' not 'bz2'. [default:
%default]")
    parser.add_option('--all-dat-files',dest='justbig',
                      default=True,
                      action='store_false',
                      help="By default, we will only convert the all_atom_correlmat
files.  If you use this option, we will convert all dat files.  Don't forget, though,
that you'll still have to call this command twice if you have some files that are bzipped
and some that are not.",
                      )

    options,args = parser.parse_args()
    if options.justbig:
        pattern =
os.path.join(options.dir,options.structurename+'_NS*_all_atom_correlmat*.dat'+options.com
pression)
    else:
        pattern =
os.path.join(options.dir,options.structurename+'_NS*.dat'+options.compression)
    fnames = glob.glob(pattern)
    print "I will convert these files",pattern,":",fnames
    for fname in fnames:
        if fname.endswith('bz2'):
            head = os.path.splitext(fname)[0]
        else:
            head = fname
        numpy_ver = head+'.numpy'
        numpybz_ver = head+'.numpy.bz2'
        if os.path.exists(numpy_ver) or os.path.exists(numpybz_ver):
            print "Skipping",fname
            continue
        if fname.endswith('bz2'):
            print "Doing",fname,numpybz_ver
            data=scipy.io.read_array(bz2.BZ2File(fname))
            data.tofile(numpy_ver)
            os.system('bzip2 %s'%numpy_ver)
        else:
            print "Doing",fname,numpy_ver
            data=scipy.io.read_array(file(fname))
            data.tofile(numpy_ver)
        del data
```

drivers/display_bridging_interactions.py

```
#!/usr/bin/env python

import sys,os

if sys.platform == 'darwin':
    PYPAT_CODE_DIR = '/Users/mglerner/work/PyPAT/code/'
elif sys.platform == 'linux2':
    PYPAT_CODE_DIR = '/users/mlerner/work/src/PyPAT/code/'

sys.path.append(PYPAT_CODE_DIR)

if __name__ == '__main__':
    from pypat.hbond import pymol_hbond_analysis
    from pypat import tool_utils

    from optparse import OptionParser

    parser = OptionParser(option_class=tool_utils.MyOption)

    parser.add_option("--name",'-n',dest="name",default="nrna",
                      help="Trajectory and topology must be named name.trj and name.top
respectively. [default: %default]"
                      )
    parser.add_option("--timestep",'-t',dest='timestep',default=5,
                      help="trajectory timestep in picoseconds. [default: %default]",
                      type='int',
                      )
    parser.add_option('--min-dwell-time','-m',dest='minrequireddwelltime',default=3,
                      help='minimum required dwell time. [default: %default]',
                      type='int',
                      )
    parser.add_option('--looseness','-l',dest='looseness',default=2,
                      help='looseness. [default: %default]',
                      type='int',
                      )
    parser.add_option("--dist-cutoff",'-d',dest="distcutoff",default=3.5,
                      help="Heavy atom to heavy atom distance cutoff. [default:
%default]",
                      type='float',
                      )
    parser.add_option('--angle-cutoff','-a',dest='anglecutoff',default=0.0,
                      help="Angle cutoff. If heavy:hydro:heavy angle must be greater than
this. [default: %default]",
                      type='float',
                      )
    parser.add_option('--min-occ','-o',dest='minocc',default=0.4,
                      help="Minimum percentage of the trajectory for which this
interaction must be occupied. [default: %default]",
                      type='float',
                      )
    parser.add_option('--dir','-r',dest='dir',default='.',
                      help="Directory in which the name_hbond_*_*.txt files are located.
[default: %default]",
                      )
    parser.add_option('--resi-criteria','-R',dest='resi_criteria',default=None,
                      help="Restrict the output to BWIs where at least one side involves
a residue in this list. [default: %default]",
                      type='onebasedintlist',
                      )

    options,args = parser.parse_args()




    import glob
    #d = '/Users/mglerner/work/Dynamics-
DHFR/MD_Files/BridgingWaterOutput/3.5A_HbridgeFiles/'
    d = options.dir
    struct = options.name
    fnames = glob.glob(os.path.join(d,'%s_hbond_?_?.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_?_??.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_??_??.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_??_???.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_???_???.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_???_????.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_????_????.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_?????_?????.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_??????_??????.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_???????_???????.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_????????_????????.txt'%struct))
    fnames += glob.glob(os.path.join(d,'%s_hbond_?????????_?????????.txt'%struct))



    a = pymol_hbond_analysis.get_hbond_trajectories(structure=options.name,
                                                    timestep=options.timestep,
                                                    fnames=fnames,
                                                    combination_method='loose',

min_required_dwell_time=options.minrequireddwelltime,
                                                    looseness=options.looseness,
                                                    dist_cutoff=options.distcutoff,
                                                    angle_cutoff=options.anglecutoff,
                                                    ) # skip the last one because we're
probably writing to it right now
    #a = get_hbond_trajectories('1rx1',5,fnames1[:50]) # skip the last one because we're
probably writing to it right now

    #nap_including_resis = list((160,)) # NAP
    #m20_including_resis = list(range(9,25)) # m20
    #tunnel_including_resis = list((5,7,8,30,33,34,92,111,112,113,114,137,153,155)) #
tunnel
    #tunnel_surface_including_resis =  list((30,33,111,137,153,155)) # tunnel surface
    #no_including_resis = None
    #including_resis = tunnel_including_resis + nap_including_resis
    #including_resis = no_including_resis
    print a.get_trajectory_string(#minocc=0.05,
                                  #minocc=0.60,
                                  minocc=options.minocc,
                                  numchunks=50,
                                  including_resis=options.resi_criteria,
                                  #sort_by='average_dwell_time',
                                  sort_by='occ',
                                  )
```

drivers/do_correlated_md_analysis.py

```
#!/usr/bin/env python

import sys,os

if sys.platform == 'darwin':
    PYPAT_CODE_DIR = '/Users/mglerner/work/Dynamics-DHFR/'
elif sys.platform == 'linux2':
    PYPAT_CODE_DIR = '/users/mlerner/work/src/Dynamics-DHFR/'

sys.path.append(PYPAT_CODE_DIR)


if __name__ == '__main__':
    from pypat.md_analysis_utils import write_max_min_ca_resi_versions
    from pypat import tool_utils
    from optparse import OptionParser
    #
    # Standard procedure for writing out the specific correlation and covariance
    # matrices that we want.
    #

    parser = OptionParser(option_class=tool_utils.MyOption,usage=tool_utils.usage)


    tool_utils.add_standard_options(parser)
    tool_utils.add_window_options(parser)

    parser.add_option("--non-ca-resis",dest="non_ca_resis",type="zerobasedintlist",
                      default=[],
                      help="Comma separated list of residues that don't contain alpha
carbons.  We need this to make some of our output images.  [default: %default], but you
could say 160,161 for example")


    parser.add_option('--single-time',dest='singletime',
                      default=None,
                      help="If you specify a single time here, we will ignore the
windowing options and perform our calculations on only one file.  The time should be
specified in a format similar to NS02.50."
                      )
    options,args = parser.parse_args()

    if options.singletime is None:
        desired=tool_utils.get_desired(options)
        for times in [i[-1] for i in desired]:
            sys.stdout.write('doing %s %s'%(times,options.structurename))
            sys.stdout.flush()

write_max_min_ca_resi_versions(os.path.join(options.outputdir,options.structurename,'%s_%
s_all_atom_correlmat.dat'%(options.structurename,times)),

os.path.join(options.outputdir,options.structurename+'_ref.pdb.1'),
                                           non_ca_resis=[i+1 for i in
options.non_ca_resis],
                                           )
    else:
```

```
write_max_min_ca_resi_versions(os.path.join(options.outputdir,options.structurename,'%s_%
s_all_atom_correlmat.dat'%(options.structurename,options.singletime)),

os.path.join(options.outputdir,options.structurename+'_ref.pdb.1'),
                                           non_ca_resis=[i+1 for i in
options.non_ca_resis],
                                           )
```

drivers/make_correlated_dynamics_plots.py

```
#!/usr/bin/env python

'''
make all of our correl and covar plots with scipy.
'''

import sys,os

if sys.platform == 'darwin':
    PYPAT_CODE_DIR = '/Users/mglerner/work/PyPAT/code'
elif sys.platform == 'linux2':
    PYPAT_CODE_DIR = '/users/mlerner/work/src/PyPAT/code'

sys.path.append(PYPAT_CODE_DIR)

if __name__ == '__main__':
    from pypat.plotting import make_correl_plots_for_movie, make_one_correl_plot
    from pypat import tool_utils
    from optparse import OptionParser
    import pylab

    usage = tool_utils.usage + """
This will spit out an html file that will show you your images.  If you
need to look at the images on another machine, tar up the html file and
output-dir/images together and move that to the other machine.
"""
    parser = OptionParser(option_class=tool_utils.MyOption,usage=usage)

    tool_utils.add_standard_options(parser)
    tool_utils.add_window_options(parser)

    parser.add_option("--cmap",dest="cmap",default="Normal",
                      help="Color map to use when making the plots. Our custom cmaps are
Normal and Scaled.  Standard matplotlib cmaps %s are also supported."%[m for m in
pylab.cm.datad.keys() if not m.endswith("_r")] + "[default: %default]",
                      )

    parser.add_option('--plot-types',dest="plottypes",
                      default='ca avg max min abs straight mainheavy allheavy
sidechainhbond hbond'.split(),
                      type="strlist",
                      help="Comma-separated list of plot types.  [default: %default]")

    parser.add_option('--single-time',dest='singletime',
                      default=None,
                      help="If you specify a single time here, we will ignore the
windowing options and display an interactive plot of the time you specify.  The time
should be, e.g., NS0.5",
                      )
    parser.add_option('--mark-resis',dest='markresis',
                      default=[],
                      type="zerobasedintlist",
                      help="A list of residues to mark on the plots.  [default:
%default]")
    parser.add_option('--highlight',dest='highlight',
                      default=0.2,
                      type='float',
                      help="How strongly to highlight the marked residues.  Note that --
highlight-mode tells us how exactly we will do the highlighting.  0.1 and 0.2 are decent
```

```
values if you want to use this feature for most plots, although you'll need something
stronger for the absolute value plots. [default: %default]")
    parser.add_option('--highlight-mode',dest='highlightmode',
                      default='positive',
                      help="When highlight-mode is 'negative' we put a white block down
on top of the marked residues, the opacity of which is controled by --highlight.  When
it's 'positive', we put that white block down on squares of residues that are *not*
highlighted instead.  When it's 'supernegative', we do just like 'negative' except that
the block will be twice as opaque where the highlighted rows and columns intersect.
Positive and supernegative seem to be more useful than negative.  [default: %default]")
    parser.add_option('--skip-resis',dest='skipresis',
                      default=[],
                      #
                      # Note to programmers: setup_ca expects this to come in as 1-based.
                      #
                      type="onebasedintlist",
                      help="A list of residues that will be skipped in the plots.
[default: %default]")
    parser.add_option('--no-ticks',dest='ticks',
                      default=True,
                      action='store_false',
                      help="do not include tick marks on the axes",
                      )
    parser.add_option('--dpi',dest='dpi',
                      default=200,
                      type='int',
                      help='dpi for figures [default: %default]',
                      )
    parser.add_option('--title',dest='title',
                      default=None,
                      help='If no title is specified, one will be automatically
generated. Note that the title is part of the filename that we save. [default:
%default]',
                      )

    options,args = parser.parse_args()
    desired=tool_utils.get_desired(options)

    #
    # If you want to make all of the plots for a given time and you want to
    # write them out to files, use this:
    if options.singletime is None:
        all_times = [i[-1] for i in desired]
        make_correl_plots_for_movie(structures=[options.structurename,],
                                    all_times=all_times,
                                    cmaps=[options.cmap,],
                                    detail_levels=['fine',],
                                    plot_types=options.plottypes,
                                    overwrite=True,
                                    image_dir=os.path.join(options.outputdir,'images'),
                                    dat_dir=options.outputdir,

ref_pdb_fname=os.path.join(options.outputdir,options.structurename+'_ref.pdb.1'),
                                    mark_resis=options.markresis,
                                    highlight=options.highlight,
                                    highlight_mode=options.highlightmode,
                                    skip_resis=options.skipresis,
                                    ticks=options.ticks,
                                    dpi=options.dpi,
                                    title=options.title,
                                    )
    #
    # If you just want to make one plot and you want it to show up in a window,
    # use this:
    #
    else:
        if len(options.plottypes) != 1:
            sys.exit("In single-plot mode, you must specify exactly one plot type.")
        print "title",options.title
        make_one_correl_plot(struct=options.structurename,
                             times=options.singletime,
                             cmap=options.cmap,
                             detail_level='fine',
                             plot_type=options.plottypes[0],
                             save_fig=False,
                             overwrite=True,
                             dat_dir=options.outputdir,
                             out_dir=os.path.join(options.outputdir,'images'),

ref_pdb_fname=os.path.join(options.outputdir,options.structurename+'_ref.pdb.1'),
                             mark_resis=options.markresis,
                             highlight=options.highlight,
                             highlight_mode=options.highlightmode,
                             skip_resis=options.skipresis,
                             ticks=options.ticks,
                             dpi=options.dpi,
                             title=options.title,
                             )
```

drivers/make_movies.py

```
#!/usr/bin/env python

import sys,os,glob

if sys.platform == 'darwin':
    PYPAT_CODE_DIR = '/Users/mglerner/work/PyPAT/code/'
elif sys.platform == 'linux2':
    PYPAT_CODE_DIR = '/users/mlerner/work/src/PyPAT/code/'

sys.path.append(PYPAT_CODE_DIR)




if __name__ == '__main__':
    from pypat import tool_utils
    import glob
    from optparse import OptionParser

    usage = tool_utils.usage + """
Please also make sure that the imagemagick utility convert
is installed and in your path.

This will spit out an html file that will show you your images.  If you
need to look at the images on another machine, tar up the html file and
output-dir/images together and move that to the other machine.

The html file will be called <struct>BigAnimatedMovies.html.
"""
    parser = OptionParser(option_class=tool_utils.MyOption,usage=tool_utils.usage)

    tool_utils.add_standard_options(parser)
    parser.add_option('--plot-types',dest="plottypes",
                      default='ca avg max min abs straight mainheavy allheavy
sidechainhbond hbond'.split(),
                      type="strlist",
                      help="Comma-separated list of plot types.  [default: %default]",
                      )
    parser.add_option('--no-slow-movies',dest="slowmovies",
                      default=True,
                      action="store_false",
                      help="Set this if you do not want to generate the movies that have
0.5s spacing between the frames.",
                      )
    parser.add_option('--movie-link',dest="movielink",
                      default='fast',
                      help="'fast' if you want the thumbnails to link to the fast images,
anything else for the slow ones. [default: %default]",
                      )
    options,args = parser.parse_args()
    html_txt = '''
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Correlated Dynamics Movies</title>
  </head>

  <body>
    <h1>Correlated Dynamics Movies</h1>

    These are animated gifs of a sliding 1NS window throughout the MD simulation.  So,
e.g., %s 2.2NS is the correlated dynamics of the %s structure from 1.7NS to 1.7NS (the
center of the window is 1.2NS).
    <ul>
      <li><b>straight</b> every atom is shown</li>
      <li><b>mainheavy</b> mainchain heavy (nonhydrogen) atoms</li>
      <li><b>allheavy</b> all heavy (nonhydrogen) atoms</li>
      <li><b>hbond</b> NOSP vs. hydrogen atoms</li>
      <li><b>sidechainhbond</b> sidechain NOSP vs. sidechain hydrogen atoms</li>
      <li><b>ca</b> alpha carbons plotted against eachother, one per residue.  This is
the standard plot in the literature.</li>
      <li><b>abs</b> the largest absolute value for each residue-residue pair</li>
      <li><b>min</b> the minimum value for each residue-residue pair</li>
      <li><b>max</b> the maximum value for each residue-residue pair</li>
    </ul>

    The full-sized animated gifs, which you can see by clicking on the versions shown
here,
    will play more slowly than the thumbnails.  If you wish to see the speeded-up
version,
    remove the "0.5" from the filename.  E.g. look at
animated_%s_resi_mainheavy_correl.gif
    instead of animated_0.5_%s_resi_mainheavy_correl.gif.

    <table>
      <tr>
        <td><b>%s</b> (closed loop starting structure)</td>
      </tr>
      ''' % (options.structurename,
             options.structurename,
             options.structurename,
             options.structurename,
             options.structurename,)




    for plot_type in options.plottypes:
        # New naming conventions mean that these are zero-padded and will be in the
correct order.
        #filenames =
glob.glob(os.path.join(options.outputdir,'images',options.structurename+" NS??? resi
"+plot_type+" correl*",)) +
glob.glob(os.path.join(options.outputdir,'images',options.structurename+" NS???? resi
"+plot_type+" correl*",)) +
glob.glob(os.path.join(options.outputdir,'images',options.structurename+" NS????? resi
"+plot_type+" correl*",)) #does anyone do any 100ns+ simulations??
        filenames =
glob.glob(os.path.join(options.outputdir,'images',options.structurename+" NS* resi
"+plot_type+" correl*",))
        if not filenames:
            print "COULD NOT FIND files for",plot_type
            continue
        prog = 'convert'
        print filenames
        print
[os.path.join(options.outputdir,'images',"animated_"+options.structurename+"_resi_"+plot_
type+"_correl.gif",),]
        args = ['-loop','0',] + filenames +
[os.path.join(options.outputdir,'images',"animated_"+options.structurename+"_resi_"+plot_
type+"_correl.gif",),]
        tool_utils.run(prog,args,verbose=True)
        prog,args = 'convert',('-resize','256x',

os.path.join(options.outputdir,'images',"animated_"+options.structurename+"_resi_"+plot_t
ype+"_correl.gif",),

os.path.join(options.outputdir,'images',"animated_"+options.structurename+"_resi_"+plot_t
ype+"_correl_thumb.gif",),
                               )
        tool_utils.run(prog,args,verbose=True)
        if options.slowmovies:
            prog,args = 'convert',('-delay','50',

os.path.join(options.outputdir,'images',"animated_"+options.structurename+"_resi_"+plot_t
ype+"_correl.gif",),

os.path.join(options.outputdir,'images',"animated_0.5_"+options.structurename+"_resi_"+pl
ot_type+"_correl.gif",),
                                   )
        tool_utils.run(prog,args,verbose=True)

        if (options.movielink == 'fast') or not options.slowmovies:
            html_txt += '''      <tr>
            <td><a href="images/animated_%s_resi_%s_correl.gif"><img
src="images/animated_%s_resi_%s_correl_thumb.gif/"/></a></td>
          </tr>\n'''%(
                      options.structurename,
                      plot_type,

                      options.structurename,
                      plot_type,
                      )
        else:
            html_txt += '''      <tr>
            <td><a href="images/animated_0.5_%s_resi_%s_correl.gif"><img
src="images/animated_%s_resi_%s_correl_thumb.gif/"/></a></td>
          </tr>\n'''%(
                      options.structurename,
                      plot_type,

                      options.structurename,
                      plot_type,
                      )

    html_txt += '''    </table>
    <hr>
  </body>
</html>
'''
    f =
file(os.path.join(options.outputdir,options.structurename+'BigAnimatedMovies.html'),'w')
    f.write(html_txt)
    f.close()
```

drivers/parse_sander_output.py

```
#!/usr/bin/env python

import sys,os,re

#
# Each section that we care about looks like this:
#
exampleSection = """
 NSTEP =     0  TIME(PS) =    0.000  TEMP(K) =   457.87  PRESS =      0.00
 Etot   = -119068.4556  EKtot   =   33623.6366  EPtot      = -152692.0922
 BOND   =     130.2398  ANGLE   =     448.8823  DIHED      =     849.5638
 1-4 NB =     515.8140  1-4 EEL =    6193.3324  VDWAALS    =   15714.9288
 EELEC  = -176544.8532  EHBOND  =       0.0000  CONSTRAINT =       0.0000
 Ewald error estimate:   0.6201E-04
 ------------------------------------------------------------------------------

===============================================================================
                      NMR restraints for step      0
 Energy (this step): Bond =     0.000   Angle =     0.000   Torsion =     0.000
 Energy (tot. run) : Bond =     0.000   Angle =     0.000   Torsion =     0.000

 DEVIATIONS:    Target=(r2+r3)/2                 Target = closer of r2/r3
            This step         Entire run        This step        Entire run
           ave.    rms       ave.    rms       ave.    rms      ave.    rms
 Bond      0.000   0.000     0.000   0.000     0.000   0.000     0.000   0.000
 Angle     0.000   0.000     0.000   0.000     0.000   0.000     0.000   0.000
 Torsion   0.000   0.000     0.000   0.000     0.000   0.000     0.000   0.000
===============================================================================
"""


class StepInfo(dict):
    #
    # see the comments in __getitem__ for info about allKeys.
    # we also use it to cheat .. we know that 'EAMBER (non-constraint)' will
    # show up eventually, but maybe not in the first step, so we'll seed our
    # keys list.
    #
    allKeys = {'EAMBER (non-constraint)':True}
    def __init__(self, inString):
        #
        # The useful lines are all of the form
        # A = B  C=D E = F
        #
        # That is, it's always name=value pairs, but
        #  - there may or may not be spaces around either
        #    side of the =.
        #  - the name may be one or more words
        #
        # We assume that we've only been passed useful lines.
        #
        # So, we need to be a little clever.  We concatenate the
        # lines into one big string.  Then we split it based on
        # equals signs.  Everything in between the equals signs
        # then gets split up based on whitespace.  So, the name-value
        # pairs are [everything but the first thing in group i-1]-
        # [first thing in group i].  We have to take care to get
        # the first and last groups correct.
        #
        inString = ' '.join(inString.split()) # get rid of spurious whitespace
        parts = inString.split('=')
        names = [parts[0].strip()]
        values = []
        for part in parts[1:-1]:
            try:
                values.append(float(part.split()[0]))
            except ValueError:
                values.append(0.0)
            names.append(' '.join(part.split()[1:]))
        values.append(float(parts[-1]))
        if len(names) != len(values):
            sys.exit('oops')
        for name,value in zip(names,values):
            self[name] = value
    def __setitem__(self,name,value):
        self.allKeys[name] = True
        return super(StepInfo,self).__setitem__(name,value)
    def __getitem__(self,item):
        #
        # If it's a "valid key" (meaning that some StepInfo has seen it before),
        # and we haven't seen it, we'll return the empty string.
        #
        if item in self.allKeys:
            val = ''
            try:
                val = super(StepInfo,self).__getitem__(item)
            except KeyError:
                pass
        else:
            val = super(StepInfo,self).__getitem__(item)
        return val
    def keys(self):
        return self.allKeys.keys()
    def iteritems(self):
        #
        # we need to make sure that this includes the fake keys in self.allKeys
        #
        for k in self.keys():
            yield k,self[k]


def steps(fileobj):
    #
    # When we get to the RMS fluctuations, we're done.
    #
    step = []
    for line in fileobj:
        if line.startswith('      R M S  F L U C T U A T I O N S'):
            print "BREAKING"
            break
        if line.startswith(' NSTEP'):
            if step:
                yield ''.join(step)
            step = [line]
        else:
            #
            # Only include lines that we know should be included
            #
            goodStarts = (' NSTEP', ' Etot', ' BOND', ' 1-4 NB', ' EELEC', ' EKCMT','
Density',  ' EAMBER (non-constraint)',)
            #goodStarts = (' NSTEP', ' Etot', ' BOND', ' 1-4 NB', ' EELEC', ' EKCMT','
Density',)
            #
            # Only some lines have EAMBER .. StepInfo deals with that.
            #
            for start in goodStarts:
                if line.startswith(start):
                    step.append(line)
    if step:
        yield(''.join(step))


def getSteps(filename):
    f = file(filename)
    for step in steps(f):
        yield StepInfo(step)
    f.close()

def getFilename(s,extension = '.txt',dir=''):
    """everything that's not a letter or number turns into an underscore"""
    return os.path.join(dir,re.sub('\W','_',s) + extension)

usage = """
usage:
parseSanderOut.py file.out dirname

will put the data in file.out into nice files in dirname.
The directory called dirname must not exist when this script is run.
Among those files are:

  data/allout.txt      tab-delimited text file with all information
                       (suitable for gnumeric, koffice or excel)
  data/Etot.txt, etc.  individual files with different types of sander
                       output (two columns per file, one is time(ps))
  results.html         html file showing you lots graphs of your data
  images/*             postscript and gif graphs of your data

So, for the most part, you probably just want to point your favorite
webbrowser at dirname/results.html.
```

```
NOTE: this script assumes that you have gnuplot and convert installed
and in your path.  This is true on the SGIs, for example, but not on garlic.

"""

if __name__ == '__main__':
    if len(sys.argv) != 3:
        print "****************************************"
        print
        print "Not enough arguments on the command line"
        print
        print "****************************************"
        print
        sys.exit(usage)
    datafileName, dir = sys.argv[1:]
    try:
        f = file(datafileName)
        f.close()
    except:
        print "****************************************"
        print
        print "Could not read %s." % datafileName
        print
        print "****************************************"
        print
        sys.exit(usage)
    if os.path.exists(dir):
        print "****************************************"
        print
        print "%s exists.  Please (re)move it or specify" % dir
        print "a different destination directory."
        print
        print "****************************************"
        print
        sys.exit(usage)

    os.system('mkdir %s' % dir)
    #
    # We write out alldata.txt, which contains all of the data.
    # We also write out <thing>.txt where <thing> is all of the
    # other types of data, e.g. EELEC.data.
    #
    allout = file(os.path.join(dir,'alldata.txt'),'w')
    # fileMap maps data type (EELEC) to file object and filename.
    # e.g. fileMap = {'EELEC':{'filename':'EELEC.txt','file':<fileobj>},...}
    #
    fileMap = {}
    #
    # We'll set the keys with the first step and reuse them
    # for all subsequent setps.  This guarantees that we have a consistent
    # number of columns.
    #
    first = True
    nonTimeKeys = None
    for step in getSteps(datafileName):
        #
        # We want TIME(PS) to be the first column
        #
        if nonTimeKeys is None:
            nonTimeKeys = step.keys()
            try:
                nonTimeKeys.remove('TIME(PS)')
            except:
                sys.exit('Could not find time(ps) in ' + str(nonTimeKeys))
        if first:
            first = False
            #
            # Set up fileMap
            #
            for key in nonTimeKeys:
                filename = getFilename(key)
                fileMap[key] = {'filename':filename,
                                'file':file(os.path.join(dir,filename),'w'),}
            #
            # Write the headers
            #
            allout.write('# TIME(PS)\t' + '\t'.join(nonTimeKeys) + '\n')
            for key in fileMap.keys():
                fileMap[key]['file'].write('# TIME(PS)\t' + key + '\n')

        #
        # line keeps track of the line for allout.txt
        #
        line = '%s\t' % step['TIME(PS)']
        for k in nonTimeKeys:
            if k != 'TIME(PS)':
                line += '%s\t' % step[k]
                fileMap[k]['file'].write('%s\t%s\n' % (step['TIME(PS)'],
                                                       step[k]))
        line += '\n'
        allout.write(line)
    #
    # Close all of the files
    #
    allout.close()
    for k in fileMap.keys():
        fileMap[k]['file'].close()

    #
    # Now we use gnuplot to make a bunch of images
    #
    imgDir = os.path.join(dir,'images')
    os.system('mkdir -p ' + imgDir)
    gpltCmd = 'set terminal postscript\n'
    for k in fileMap.keys():
        imgName = getFilename(k,'.ps',imgDir)
        gpltCmd += "set output '%s'\n" % imgName
        gpltCmd += "plot '%s'\n" % os.path.join(dir,fileMap[k]['filename'])
    gpltFilename = os.path.join(dir,'commands.gnuplot')
    gpltFile = file(gpltFilename,'w')
    gpltFile.write(gpltCmd)
    gpltFile.close()
    os.system('/usr/bin/env gnuplot %s' % gpltFilename)
    #
    # gnuplot can't always make gifs, so we make postscript files and convert
    # them into gifs.
    #
    thumbPart = '-size 200x200 -geometry 200x200'
    for k in fileMap.keys():
        ps = getFilename(k,'.ps',imgDir)
        gif = getFilename(k,'.gif',imgDir)
        thumb = getFilename(k,'_thumb.gif',imgDir)
        os.system('/usr/bin/env convert -rotate 90 %s %s' % (ps,gif))
        os.system('/usr/bin/env convert -rotate 90 %s %s %s' % (thumbPart,
                                                                ps,
                                                                thumb))

    #
    # This will be a really simple html file
    #
    numCols = 4
    htmlFile = file(os.path.join(dir,'results.html'),'w')
    htmlFile.write("<html><head></head><body>")
    htmlFile.write('<table><tr><td colspan="%s">These are the thumbnails .. click on them
for the big picture(s)</td>' % numCols)
    count = 0
    for k in fileMap.keys():
        if count % numCols == 0:
            htmlFile.write('</tr><tr>')
        count += 1
        bigName = getFilename(k,'.gif','images')
        thumbName = getFilename(k,'_thumb.gif','images')
        htmlFile.write('<td>%s<br>'%(k))
        htmlFile.write('<a href="#%s"><img src="%s"></a>'%(k,thumbName))
        htmlFile.write('</td>\n')
    htmlFile.write('</tr></table>\n')
    for k in fileMap.keys():
        bigName = getFilename(k,'.gif','images')
        htmlFile.write('<br>')
        htmlFile.write('<a name="%s"><a href="%s"><img src="%s"></a></a>\n' %
(k,getFilename(k,'.txt'), bigName))
    htmlFile.write("</body></html>\n")
    htmlFile.close()
```

drivers/run_ptraj.py

```
#!/usr/bin/env python

import sys,os

if sys.platform == 'darwin':
    PYPAT_CODE_DIR = '/Users/mglerner/work/Dynamics-DHFR/'
elif sys.platform == 'linux2':
    PYPAT_CODE_DIR = '/users/mlerner/work/src/Dynamics-DHFR/'

sys.path.append(PYPAT_CODE_DIR)

if __name__ == '__main__':
    from pypat import tool_utils
    import glob
    from optparse import OptionParser

    usage = tool_utils.usage + """
Please also make sure that ptraj is installed and in your path.

This mimics the following shell script:

    export PTRAJ_INPUT_DIR=/users/mlerner/tmp/spronk/ptraj_files/
    export PRMTOP=/users/spronk/temp/1sgz/1sgz.nowat.prmtop
    for f in $PTRAJ_INPUT_DIR/*.ptraj; do ptraj $PRMTOP <$f > $f.out; done
"""
    parser = OptionParser(usage=tool_utils.usage)

    tool_utils.add_standard_options(parser)

    parser.add_option("--input-dir",dest="inputdir",
                      default="./",
                      help="Directory that contains our input files.  It should contain
the prmtop file and the mdcrd file.  [default: %default]")
    parser.add_option("--prmtop",dest="prmtop",
                      default="1sgz.nowat.prmtop",
                      help="Name of prmtop file")

    options,args = parser.parse_args()

    filenames =
glob.glob(os.path.join(options.outputdir,'calculate_%s*_correl_and_covar.ptraj'%options.s
tructurename))
    filenames +=
glob.glob(os.path.join(options.outputdir,'write_%s_ref_pdb.ptraj'%options.structurename))
    print "I found these ptraj files to run:",filenames
    for f in filenames:
        prog,args = 'ptraj',(os.path.join(options.inputdir,options.prmtop), f)
        retcode,progout = tool_utils.run(prog,args)
        outf = file(f+'.out','w')
        outf.write(progout)
        outf.close()
        print "Ran",prog,args,"with result",retcode
```

drivers/write_ptraj_input_files.py

```
#!/usr/bin/env python
import os,sys
import sys,os

if sys.platform == 'darwin':
    PYPAT_CODE_DIR = '/Users/mglerner/work/Dynamics-DHFR/'
elif sys.platform == 'linux2':
    PYPAT_CODE_DIR = '/users/mlerner/work/src/Dynamics-DHFR/'

sys.path.append(PYPAT_CODE_DIR)

if __name__ == '__main__':
    from pypat.runningptraj.write_ptraj_input_files import write_ptraj_input_files
    from pypat import tool_utils
    from optparse import OptionParser

    usage = tool_utils.usage + """
This script will emit the PDB file that you will use as a
reference structure later on.  It will live in <outputdir>/<structure>_ref.pdb.1
"""

    parser = OptionParser(option_class=tool_utils.MyOption,usage=usage)

    tool_utils.add_standard_options(parser)
    tool_utils.add_window_options(parser)

    parser.add_option("--input-dir",dest="inputdir",
                      default="./",
                      help="Directory that contains our input files.  It should contain
the prmtop file and the mdcrd file.  [default: %default]")

    parser.add_option("--mdcrd",dest="mdcrd",
                      type="strlist",
                      default=["1sgz.nowat.0-6.mdcrd",],
                      help="Comma-separated list of mdcrd files")

    parser.add_option("--ps",dest="ps",
                      default=5,
                      type="int",
                      help="Number of ps per frame.  [default: %default]")

    parser.add_option("--align",dest="align",
                      default="all",
                      help="How to align. 'all' means 'rms first *'.  'none' means no
alignment.  Any other string will be treated as the alignment string.  For example, if
you say ':1-428@CA' the ptraj file will say 'rms first :1-428@CA'.  [default: %default]",
                      )
    parser.add_option("--strip-hydros",dest="strip_hydros",
                      default=False,
                      action="store_true",
                      help="Strip the hydrogens out during the ptraj runs.",
                      )
    parser.add_option("--strip-waters",dest="strip_waters",
                      default=False,
                      action="store_true",
                      help="Strip the waters out during the ptraj runs.  This assumes
they're named WAT.",
                      )
    parser.add_option("--write-covar",dest="write_covar",
                      default=False,
                      action="store_true",
                      help="Write out the covariance matrix [default: %default]",
                      )
    parser.add_option("--other-ptraj-strips",dest="other_ptraj_strips",
                      default=[],
                      type="strlist",
                      help="Comma-separated list of other things that ptraj should strip.
For example, could be :WAT,:BOB and we would add two lines, one saying strip :WAT and one
saying strip :BOB."
                      )


    options,args = parser.parse_args()

    desired=tool_utils.get_desired(options)
    #
    # Build up the ptraj header.
    # 1) read in the mdcrd files
    # 2) strip things
    # 3) rms vs. first one
    #
    # VERY IMPORTANT NOTE: do the rms *after* you've done everything
    # else.  Otherwise, it tries to align things with the waters, etc.
    # and all of your correlations will be totally wrong.
    #
    ptraj_header = ''
    pdb_ptraj_header = ''
    pdb_ptraj_header += 'trajin %s 1 1
1\n\n'%os.path.join(options.inputdir,options.mdcrd[0])

    for mdcrd in options.mdcrd:
        ptraj_header += 'trajin %s\n'%os.path.join(options.inputdir,mdcrd)
    ptraj_header += '\n'

    if options.strip_hydros:
        ptraj_header += 'strip @H*\n'
        pdb_ptraj_header += 'strip @H*\n'
    if options.strip_waters:
        ptraj_header += 'strip :WAT\n'
        pdb_ptraj_header += 'strip :WAT\n'
    for strip in options.other_ptraj_strips:
        ptraj_header += 'strip %s\n'%strip
        pdb_ptraj_header += 'strip %s\n'%strip

    if options.align in 'none None NONE no NO No'.split():
        pass
    elif options.align == 'all':
        ptraj_header += 'rms first *\n'
        pdb_ptraj_header += 'rms first *\n'
    else:
        ptraj_header += 'rms first %s\n'%options.align
        pdb_ptraj_header += 'rms first %s\n'%options.align



    write_ptraj_input_files(dir_containing_ptraj_files=options.outputdir,

ptraj_output_dir=os.path.join(options.outputdir,options.structurename),
                            filename_prefix=options.structurename,
                            desired=desired,
                            ps_per_frame=options.ps,
                            ptraj_header=ptraj_header,

fname_template='calculate_'+options.structurename+'_%s_correl_and_covar.ptraj',

write_out_pdb_file=os.path.join(options.outputdir,options.structurename+'_ref.pdb'),
                            write_covar=options.write_covar,
                            pdb_ptraj_header=pdb_ptraj_header,
                            )
```

pypat/__init__.py

```
#!/usr/bin/env python
#blank
```

pypat/md_analysis_utils.py

```
#!/usr/bin/env python2.4

"""
This is not really meant to be generalizable.
But, hey, knock yourself out.
"""
```

```
from scipy import *

import sys,glob,os,bz2
from tool_utils import read_data

def setup_ca(reference_fname,indexing=1,skip_resis=[]):
    """
    most of the code in my_md_analysis assumes 1-based indexing
    for this, but make_plot wants 0-based indexing.  in the end,
    i should switch everything over to 0-based, but i'll leave
    this in as a hack for now.
    """
    master_residue_list = []
    reference_coords = []
    ca_atom_id_list = []
    skip_atom_id_list = []
    #
    # We have a few naming conventions to deal with, like
    # 1HD2 vs. HD12.  One thing that seems consistent is that
    # the first non-numeric character in the atom name tells
    # us the element.
    #
    # No need for speed here.
    #
    def is_hydro(atom_name):
        c = atom_name[0]
        if c in '1234567890':
            c = atom_name[1]
        return c in 'H'
    def is_nosp(atom_name):
        c = atom_name[0]
        if c in '1234567890':
            c = atom_name[1]
        return c in 'NOSP'
    def is_mainchain(atom_name,resn):
        return (resn in 'NAP'.split()) or (atom_name in 'CA C N O H HA'.split())


    hydro_atom_id_list = []
    nonhydro_atom_id_list = []
    nosp_atom_id_list = []
    nonnosp_atom_id_list = []
    mainchain_atom_id_list = []
    sidechain_atom_id_list = []
    mainchain_nonhydro_atom_id_list = []
    sidechain_hydro_atom_id_list = []
    sidechain_nonhydro_atom_id_list = []
    f = file(reference_fname)
    generated_atom_id = 1
    for line in f:
        parts = line.split()
        if parts[0] not in ['ATOM','HETATM']:
            continue
        #atom,atom_id,atom_name,resn,chain,resi,x,y,z,occupancy,b,elem_name = parts
        atom,atom_id,atom_name,resn,chain,resi,x,y,z = parts[:9]
        #
        # One little hack to deal with missing chain info.  If it's a number,
        # it's not really the chain.
        #
        try:
            junk = float(chain)
            atom,atom_id,atom_name,resn,resi,x,y,z = parts[:8]
        except ValueError:
            pass

        x,y,z = map(float,(x,y,z))
        try:
            resi,atom_id = map(int,(resi,atom_id))
        except ValueError:
            print "Trouble getting resi,atom_id from",resi,atom_id,line.strip()
            raise
        if resi in skip_resis:
            skip_atom_id_list.append(atom_id)
        if is_hydro(atom_name):
            hydro_atom_id_list.append(atom_id)
        else:
            nonhydro_atom_id_list.append(atom_id)

        if is_nosp(atom_name):
            nosp_atom_id_list.append(atom_id)
        else:
            nonnosp_atom_id_list.append(atom_id)

        if is_mainchain(atom_name,resn):
            mainchain_atom_id_list.append(atom_id)
            if not is_hydro(atom_name):
                mainchain_nonhydro_atom_id_list.append(atom_id)
        else:
            sidechain_atom_id_list.append(atom_id)
            if is_hydro(atom_name):
                sidechain_hydro_atom_id_list.append(atom_id)
            else:
                sidechain_nonhydro_atom_id_list.append(atom_id)

        if atom_name != 'CA':
            continue
        #
        # 1RX2 claims to model two conformations for ASP 116.  I only see one
        # in the PDB file, and it's conformation A.
        #
        if resn == 'AASP' and resi == 116 and '1RX2' in reference_fname: resn = 'ASP'
        elif resn == 'AASP': print "Unrecognized residue: AASP"
        master_residue_list.append((resi,resn))
        reference_coords.append((x,y,z))
        ca_atom_id_list.append(atom_id)
    f.close()
    results =
master_residue_list,reference_coords,ca_atom_id_list,hydro_atom_id_list,nonhydro_atom_id_
list,nosp_atom_id_list,nonnosp_atom_id_list,mainchain_atom_id_list,sidechain_atom_id_list
,mainchain_nonhydro_atom_id_list,sidechain_hydro_atom_id_list,sidechain_nonhydro_atom_id_
list,skip_atom_id_list

    #
    # Now skip the things in the skip list
    #
    _results = []
    for (_i,thing) in enumerate(results):
        if not thing:
            # empty lists
            _results.append(thing)
            continue
        if type(thing[0]) == type(()):
            # master_residue_list has (1,'MET'), etc.
            thing = [i for i in thing if i[0] not in skip_atom_id_list]
        elif type(thing[0]) == type(1):
            thing = [i for i in thing if i not in skip_atom_id_list]
        else:
            print "wha?",thing
            a = 1/0
        _results.append(thing)
    results = _results


    #
    # And now take care of the indexing (0 or 1) problem
    #
    if indexing == 0:
        for thing in results:
            for i in range(len(thing)):
                if type(thing[0]) == type(()):
                    # master_residue_list has (1,'MET'), etc.
                    thing[i] = (thing[i][0] - 1,thing[i][1])
                elif type(thing[0]) == type(1):
                    thing[i] = thing[i] - 1
                else:
                    print "wha?",thing
                    a = 1/0
    return results

_parse_count = 0
def parse_ca(fname,master_residue_list):
    global _parse_count
    _parse_count += 1
    if divmod(_parse_count,100)[-1] == 0:
        sys.stdout.write('.')
        sys.stdout.flush()
    residue_list = []
    coords = []
    f = file(fname)
    for line in f:
        parts = line.split()
        if parts[0] not in ['ATOM','HETATM']:
            continue
        atom,atom_id,atom_name,resn,chain,resi,x,y,z,occupancy,b = parts
        if atom_name != 'CA':
            continue
        x,y,z = map(float,(x,y,z))
        resi,atom_id = map(int,(resi,atom_id))
        #
        # In order to compare different structures, we cheat a little bit.
        # Here, we say that all Histidine residues will be called HIS
        #
        if resn in ('HIE','HID','HIP'): resn = 'HIS'
        residue_list.append((resi,resn))
        coords.append((x,y,z))
    f.close()
    if residue_list != master_residue_list:
        #print master_residue_list
        #print residue_list
        print "broken",[(i,residue_list[i],master_residue_list[i]) for i in
range(len(residue_list)) if residue_list[i] != master_residue_list[i]]
        sys.exit()
    return coords

def parse_all(fname,master_residue_list):
    global _parse_count
    _parse_count += 1
    if divmod(_parse_count,100)[-1] == 0:
        sys.stdout.write(',')
        sys.stdout.flush()
    coords = []
    residue_list = []
    f = file(fname)
    for line in f:
        parts = line.split()
        if parts[0] not in ['ATOM','HETATM']:
            continue
        atom,atom_id,atom_name,resn,chain,resi,x,y,z,occupancy,b = parts
        x,y,z = map(float,(x,y,z))
        resi,atom_id = map(int,(resi,atom_id))
        #
        # In order to compare different structures, we cheat a little bit.
        # Here, we say that all Histidine residues will be called HIS
        #
        if resn in ('HIE','HID','HIP'): resn = 'HIS'
        if atom_name == 'CA':
            residue_list.append((resi,resn))
        coords.append((x,y,z))
    f.close()
    if residue_list != master_residue_list:
        #print master_residue_list
        #print residue_list
        print "broken",[(i,residue_list[i],master_residue_list[i]) for i in
range(len(residue_list)) if residue_list[i] != master_residue_list[i]]
        sys.exit()
    return coords


vds_radii = {'H':1.20,
             'C':1.7,
             'N':1.55,
             'O':1.52,
             'F':1.35,
             'P':1.9,
             'S':1.85,
             'Cl':1.8,
             }
def nap16O7phe31min_dist(s,include_hydro):
    '''
    nap 160 O7N is
ATOM   2499  O7N NAP A 160      31.906  43.045  14.292  0.00  0.00
    phe31 is these:
ATOM    464  N   PHE A  31      29.869  36.434   7.846  0.00  0.00
ATOM    465  H   PHE A  31      30.728  36.378   8.374  0.00  0.00
ATOM    466  CA  PHE A  31      29.490  37.827   7.596  0.00  0.00
ATOM    467  HA  PHE A  31      28.684  38.052   8.295  0.00  0.00
ATOM    468  CB  PHE A  31      30.654  38.827   7.849  0.00  0.00
ATOM    469  HB2 PHE A  31      31.411  38.440   7.165  0.00  0.00
ATOM    470  HB3 PHE A  31      31.011  38.747   8.876  0.00  0.00
ATOM    471  CG  PHE A  31      30.331  40.229   7.525  0.00  0.00
ATOM    472  CD1 PHE A  31      29.332  40.919   8.262  0.00  0.00
ATOM    473  HD1 PHE A  31      28.856  40.455   9.113  0.00  0.00
ATOM    474  CE1 PHE A  31      28.974  42.223   7.928  0.00  0.00
ATOM    475  HE1 PHE A  31      28.310  42.806   8.549  0.00  0.00
ATOM    476  CZ  PHE A  31      29.667  42.830   6.822  0.00  0.00
ATOM    477  HZ  PHE A  31      29.388  43.825   6.505  0.00  0.00
ATOM    478  CE2 PHE A  31      30.522  42.121   5.987  0.00  0.00
ATOM    479  HE2 PHE A  31      31.146  42.680   5.304  0.00  0.00
ATOM    480  CD2 PHE A  31      30.872  40.855   6.386  0.00  0.00
ATOM    481  HD2 PHE A  31      31.673  40.345   5.873  0.00  0.00
ATOM    482  C   PHE A  31      28.903  37.987   6.204  0.00  0.00
ATOM    483  O   PHE A  31      27.702  38.399   6.116  0.00  0.00
    '''
    phe_nonhydro_ids = [i - 1 for i in [464,466,468,471,472,474,476,478,480,482,483]]
    phe_hydro_ids = [i - 1 for i in [465,467,469,470,473,475,477,479,481]]
    nap_id = 2499 - 1
    if include_hydro:
        phes = phe_nonhydro_ids + phe_hydro_ids
    else:
        phes = phe_nonhydro_ids

    dists = []
    for phe in phes:
        dists.append(((s[phe][0] - s[nap_id][0])**2 + (s[phe][1] - s[nap_id][1])**2 +
(s[phe][2] - s[nap_id][2])**2)**0.5)
    return min(dists)

def ile50leu28min_dist(s,include_hydro,vdw):
    '''
    ile is residue 50, which corresponds to these:
ATOM    789  N   ILE A  50      36.039  51.305   7.008  0.00  0.00
ATOM    790  H   ILE A  50      35.197  51.604   7.478  0.00  0.00
ATOM    791  CA  ILE A  50      35.920  50.306   5.913  0.00  0.00
ATOM    792  HA  ILE A  50      36.626  49.526   6.197  0.00  0.00
ATOM    793  CB  ILE A  50      34.548  49.572   5.812  0.00  0.00
ATOM    794  HB  ILE A  50      34.634  48.734   5.122  0.00  0.00
ATOM    795  CG2 ILE A  50      34.338  48.759   7.152  0.00  0.00
ATOM    796 1HG2 ILE A  50      33.464  48.123   7.010  0.00  0.00
ATOM    797 2HG2 ILE A  50      35.292  48.296   7.399  0.00  0.00
ATOM    798 3HG2 ILE A  50      33.946  49.490   7.859  0.00  0.00
ATOM    799  CG1 ILE A  50      33.333  50.451   5.455  0.00  0.00
ATOM    800 2HG1 ILE A  50      33.017  51.019   6.330  0.00  0.00
ATOM    801 3HG1 ILE A  50      33.693  51.294   4.863  0.00  0.00
ATOM    802  CD1 ILE A  50      32.267  49.782   4.626  0.00  0.00
ATOM    803 1HD1 ILE A  50      31.563  50.542   4.285  0.00  0.00
ATOM    804 2HD1 ILE A  50      31.713  49.138   5.308  0.00  0.00
ATOM    805 3HD1 ILE A  50      32.795  49.237   3.843  0.00  0.00
ATOM    806  C   ILE A  50      36.481  50.726   4.555  0.00  0.00
ATOM    807  O   ILE A  50      36.965  49.871   3.850  0.00  0.00
    leu28 is these:
ATOM    411  N   LEU A  28      34.634  36.196   7.999  0.00  0.00
ATOM    412  H   LEU A  28      35.557  35.797   8.082  0.00  0.00
ATOM    413  CA  LEU A  28      34.258  36.995   6.836  0.00  0.00
ATOM    414  HA  LEU A  28      33.750  37.861   7.263  0.00  0.00
ATOM    415  CB  LEU A  28      35.554  37.350   6.057  0.00  0.00
ATOM    416  HB2 LEU A  28      35.988  36.377   5.826  0.00  0.00
ATOM    417  HB3 LEU A  28      35.319  37.721   5.059  0.00  0.00
ATOM    418  CG  LEU A  28      36.585  38.145   6.693  0.00  0.00
ATOM    419  HG  LEU A  28      36.899  37.709   7.641  0.00  0.00
ATOM    420  CD1 LEU A  28      37.845  38.434   5.827  0.00  0.00
ATOM    421 1HD1 LEU A  28      38.434  39.296   6.138  0.00  0.00
ATOM    422 2HD1 LEU A  28      38.500  37.569   5.925  0.00  0.00
ATOM    423 3HD1 LEU A  28      37.485  38.637   4.819  0.00  0.00
ATOM    424  CD2 LEU A  28      36.097  39.603   6.905  0.00  0.00
ATOM    425 1HD2 LEU A  28      35.930  40.049   5.925  0.00  0.00
ATOM    426 2HD2 LEU A  28      36.688  40.264   7.539  0.00  0.00
ATOM    427 3HD2 LEU A  28      35.166  39.533   7.469  0.00  0.00
ATOM    428  C   LEU A  28      33.324  36.222   5.961  0.00  0.00
ATOM    429  O   LEU A  28      32.443  36.867   5.383  0.00  0.00
    '''
    ile_nonhydro_ids = [i - 1 for i in [789,791,793,795,799,802,806,807]]
    ile_hydro_ids = [i - 1 for i in [790,792,794,796,797,798,800,801,803,804,805]]
    leu_nonhydro_ids = [i - 1 for i in [411,413,415,418,420,424,428,429]]
    leu_hydro_ids = [i - 1 for i in [412,414,416,417,419,421,422,423,425,426,427]]
    if include_hydro:
        iles = ile_nonhydro_ids + ile_hydro_ids
        leus = leu_nonhydro_ids + leu_hydro_ids
    else:
        iles = ile_nonhydro_ids
        leus = leu_nonhydro_ids

    dists = []
    for ile in iles:
        for leu in leus:
            dists.append(((s[ile][0] - s[leu][0])**2 + (s[ile][1] - s[leu][1])**2 +
(s[ile][2] - s[leu][2])**2)**0.5)
    return min(dists)


def ile50leu28ca_dist(s):
    return ((s[49][0] - s[27][0])**2 + (s[49][1] - s[27][1])**2 + (s[49][2] -
s[27][2])**2)**0.5

def ile50asp27ca_dist(s):
    return ((s[49][0] - s[26][0])**2 + (s[49][1] - s[26][1])**2 + (s[49][2] -
s[26][2])**2)**0.5

def bynumber_dist(s,n1,n2):
    return ((s[n1][0] - s[n2][0])**2 + (s[n1][1] - s[n2][1])**2 + (s[n1][2] -
s[n2][2])**2)**0.5

def simple_rmsd(s1,s2):
    """
    rmsd = sqrt((1/N)*sum((x_n - y_n)^2))
    """
    N = len(s1)
    _range = range(N)
    deviations = [(s1[i][0] - s2[i][0])**2 +
                  (s1[i][1] - s2[i][1])**2 +
                  (s1[i][2] - s2[i][2])**2 for i in _range]
    return ((1/float(N))*sum(deviations))**0.5

def write_rmsds_and_ile50leu28ca_dists(ref_fname,
                                       target_pattern,
                                       rmsd_out_fname,
                                       ileleu_out_fname,
                                       ileleu_min_out_fname,
                                       ileleu_min_hydro_out_fname,
                                       ileleu_min_vdw_out_fname,
                                       ileleu_min_vdw_hydro_out_fname,
                                       nap160phe31_out_fname,
                                       nap160phe31_min_out_fname,
                                       nap160phe31_min_hydro_out_fname,
                                       ileasp_out_fname,
                                       out_dir):
```

```
    '''
    Wrapper function to write out all of the RMSDS and distances I care about

    target_pattern: we glob this together to get the target files
    '''
    structure_fnames = glob.glob(target_pattern)
    if not structure_fnames:
        print "I found zero files matching this pattern:",target_pattern

master_residue_list,reference_coords,ca_atom_id_list,hydro_atom_id_list,nonhydro_atom_id_
list,nosp_atom_id_list,nonnosp_atom_id_list,mainchain_atom_id_list,sidechain_atom_id_list
,mainchain_nonhydro_atom_id_list,sidechain_hydro_atom_id_list,sidechain_nonhydro_atom_id_
list,skip_atom_id_list = setup_ca(ref_fname)
    every_atom_coords = [parse_all(fname,master_residue_list) for fname in
structure_fnames]
    all_coords = [parse_ca(fname,master_residue_list) for fname in structure_fnames]
    if 1:
        # individual distances
        #
        # ILE50 LEU20 CA dists don't care about the loops, so do them first
        #


        ile50leu28min_dists = [ile50leu28min_dist(coords,include_hydro=False,vdw=False)
for coords in every_atom_coords]
        f = file(os.path.join(out_dir,ileleu_min_out_fname),'w')
        for ileleud in ile50leu28min_dists: f.write('%s\n'%ileleud)
        f.close()

        ile50leu28min_dists = [ile50leu28min_dist(coords,include_hydro=True,vdw=False)
for coords in every_atom_coords]
        f = file(os.path.join(out_dir,ileleu_min_hydro_out_fname),'w')
        for ileleud in ile50leu28min_dists: f.write('%s\n'%ileleud)
        f.close()

        ile50leu28min_dists = [ile50leu28min_dist(coords,include_hydro=False,vdw=True)
for coords in every_atom_coords]
        f = file(os.path.join(out_dir,ileleu_min_vdw_out_fname),'w')
        for ileleud in ile50leu28min_dists: f.write('%s\n'%ileleud)
        f.close()

        ile50leu28min_dists = [ile50leu28min_dist(coords,include_hydro=True,vdw=True) for
coords in every_atom_coords]
        f = file(os.path.join(out_dir,ileleu_min_vdw_hydro_out_fname),'w')
        for ileleud in ile50leu28min_dists: f.write('%s\n'%ileleud)
        f.close()

        nap160O7phe31cg_dists = [bynumber_dist(coords,2499-1,471-1) for coords in
every_atom_coords]
        f = file(os.path.join(out_dir,nap160phe31_out_fname),'w')
        for npd in nap160O7phe31cg_dists: f.write('%s\n'%npd)
        f.close()

        nap160O7phe31_min_dists = [nap16O7phe31min_dist(coords,include_hydro=False) for
coords in every_atom_coords]
        f = file(os.path.join(out_dir,nap160phe31_min_out_fname),'w')
        for npd in nap160O7phe31_min_dists: f.write('%s\n'%npd)
        f.close()

        nap160O7phe31_min_hydro_dists = [nap16O7phe31min_dist(coords,include_hydro=True)
for coords in every_atom_coords]
        f = file(os.path.join(out_dir,nap160phe31_min_hydro_out_fname),'w')
        for npd in nap160O7phe31_min_hydro_dists: f.write('%s\n'%npd)
        f.close()

        if 0:

            ile50leu28ca_dists = [ile50leu28ca_dist(coords) for coords in all_coords]
            f = file(os.path.join(out_dir,ileleu_out_fname),'w')
            for ileleud in ile50leu28ca_dists: f.write('%s\n'%ileleud)
            f.close()
            ile50asp27ca_dists = [ile50asp27ca_dist(coords) for coords in all_coords]
            f = file(os.path.join(out_dir,ileasp_out_fname),'w')
            for ileaspd in ile50asp27ca_dists: f.write('%s\n'%ileaspd)
            f.close()

    if 0:

        #
        # Now do the loops
        #
        standard_parts = 'm20 all FG CD GH noloops subdomain1 subdomain2
substrate_binding substrate_binding2'.split()
        residue_chunks = '0-20 20-40 40-60 60-80 80-100 100-120 120-140 140-160'.split()
        anti_correlated_parts = 'm20 40-50 59-68 71-74 95-97'.split() + '116-124 142-
149'.split() # 116-124 is correlated with 40-50 and 59-68, 142-149 is correlated with 40-
50
        correlated_parts = '110-114 134-141 107-113 149-156 132-142 144-157'.split() +
'115-125 1-10 58-62 43-49 38-63 42-51'.split()
        #for structure_parts in correlated_parts + anti_correlated_parts + standard_parts
+ residue_chunks:
        for structure_parts in 'substrate_binding2'.split():
            this_fname = os.path.join(out_dir,os.path.splitext(rmsd_out_fname)[0] + '_' +
structure_parts + os.path.splitext(rmsd_out_fname)[1])
            print "Doing",this_fname
            structure_parts_map = {# standard parts
                                   'all':(0,162),
                                   'm20':(8,24),
                                   'FG' :(115,132),
                                   'CD' :(63,71),
                                   'GH' :(141,150),
                                   'subdomain2':(37,106),
                                   #Subdomain 2  38-106 (Sawaya/Kraut, Fig 9B)
                                   # residue chunks
                                   '0-20':(0,20),
                                   '20-40':(20,40),
                                   '40-60':(40,60),
                                   '60-80':(60,80),
                                   '80-100':(80,100),
                                   '100-120':(100,120),
                                   '120-140':(120,140),
                                   '140-160':(140,160),
                                   # Anti-correlated parts
                                   '40-50':(39,50),
                                   '59-68':(58,68),
                                   '71-74':(70,74),
                                   '95-97':(94,97),
                                   '116-124':(115,124),
                                   '142-149':(141,149),
                                   # Correlated parts
                                   '110-114':(109,114),
                                   '134-141':(133,141),
                                   '107-113':(106,113),
                                   '149-156':(148,156),
                                   '132-142':(131,142),
                                   '144-157':(143,157),
                                   '115-125':(114,125),
                                   '1-10':(0,10),
                                   '58-62':(57,62),
                                   '43-49':(42,49),
                                   '38-63':(37,63),
                                   '42-51':(41,51),
                                   }
            if structure_parts == 'noloops':
                bad_idxs = range(*structure_parts_map['m20']) +
range(*structure_parts_map['FG']) + range(*structure_parts_map['CD']) +
range(*structure_parts_map['GH'])
                good_idxs = [i for i in range(*structure_parts_map['all']) if i not in
bad_idxs]
                print "structure_parts",structure_parts,"residues",good_idxs
                rmsds = [simple_rmsd([reference_coords[i] for i in good_idxs if i <
len(reference_coords)],[coords[i] for i in good_idxs if i < len(coords)]) for coords in
all_coords]
            elif structure_parts == 'subdomain1':
                #Subdomain 1  1-37, 107-159 (Sawaya/Kraut, Fig 9B)
                rmsds = 
[simple_rmsd(reference_coords[0:37]+reference_coords[106:159],coords[0:37]+coords[106:159
]) for coords in all_coords]
            elif structure_parts == 'substrate_binding':
                # my best guess at reproducing fig. 1 from Boehr is 27-40,50-59,6-7,112-
113
                rmsds = 
[simple_rmsd(reference_coords[26:40]+reference_coords[49:59]+reference_coords[5:7]+refere
nce_coords[111:113],

coords[26:40]+coords[49:59]+coords[5:7]+coords[111:113]) for coords in all_coords]
            elif structure_parts == 'substrate_binding2':
                # From Osborne2003, 27+36+37+54+96+5+6+26+27+28+29+36+37+50+51+52+54+57
                good_idxs = [i - 1 for i in (5,6,26,27,28,29,36,37,50,51,52,54,57,96)]
                rmsds = [simple_rmsd([reference_coords[i] for i in good_idxs],[coords[i]
for i in good_idxs]) for coords in all_coords]
            else:
                print 
"structure_parts",structure_parts,"residues",structure_parts_map[structure_parts]
                start,stop = structure_parts_map[structure_parts]
                rmsds = [simple_rmsd(reference_coords[start:stop],coords[start:stop]) for
coords in all_coords]
            f = file(this_fname,'w')
            for rmsd in rmsds: f.write('%s\n'%rmsd)
            f.close()

def get_resi_to_atom_id_map(pdb_fname,indexing=1):
    """
    most of the code in my_md_analysis assumes 1-based indexing
    for this, but make_plot wants 0-based indexing.  in the end,
    i should switch everything over to 0-based, but i'll leave
    this in as a hack for now.
    """
    atom_id_map = {}
    f = file(pdb_fname)
    generated_atom_id = 1
    for line in f:
        parts = line.split()
        if parts[0] not in ['ATOM','HETATM']:
            continue
        #atom,atom_id,atom_name,resn,chain,resi,x,y,z,occupancy,b,elem_name = parts
        atom,atom_id,atom_name,resn,chain,resi,x,y,z = parts[:9]

        #
        # One little hack to deal with missing chain info.  If it's a number,
        # it's not really the chain.
        #
        try:
            junk = float(chain)
            atom,atom_id,atom_name,resn,resi,x,y,z = parts[:8]
        except ValueError:
            pass
        try:
            resi,atom_id = map(int,(resi,atom_id))
        except ValueError:
            print "trouble with resi",resi,"atom_id",atom_id,line.strip()
            raise
        atom_id_map.setdefault(resi,[]).append(atom_id)
    f.close()
    if indexing == 0:
        aim = {}
        for k,v in atom_id_map.iteritems():
            aim[k-1] = [i - 1 for i in v]
        atom_id_map = aim
    return atom_id_map

def 
get_max_min_ca_resi_data(all_atom_data,atom_id_map,ca_atom_id_list,non_ca_resis=[160,]):
    """
    non_ca_resis is a list of resis that do not contain alpha carbons.

    returns everything as a dict.  you may run out of memory here.
    """
    abs_result = zeros((len(atom_id_map),len(atom_id_map)))
    max_result = zeros((len(atom_id_map),len(atom_id_map)))
    min_result = zeros((len(atom_id_map),len(atom_id_map)))
    ca_result  = zeros((len(atom_id_map),len(atom_id_map)))
    #nonhydro_result = zeros((len(nonhydro_atom_id_list),len(nonhydro_atom_id_list)))
    #
    # remember that atom_id and resi are 1-based, whie the array is 0-based.
    # practically, that means that we use 1-based indices when looking things
    # up in atom_id_map and 0-based indices when looking things up in
    # all_atom_data and XXX_results.
    #
    for r1 in atom_id_map:
        for r2 in atom_id_map:
            atom_ids1 = atom_id_map[r1]
            atom_ids2 = atom_id_map[r2]
            try:
                all_vals = [all_atom_data[i1-1][i2-1] for i1 in atom_ids1 for i2 in
atom_ids2]
            except IndexError:
                print "i1",i1,"i2",i2
                print "all_atom_data.keys",all_atom_data.shape
                print "all_atom_data[i1-1].keys",all_atom_data[i1-1].shape
                print "r1,r2",r1,r2
                print "atom_ids1",atom_ids1
                print "atom_ids2",atom_ids2
                raise

            if (r1 in non_ca_resis) or (r2 in non_ca_resis):
                #
                # Special case for NAP.  That's resi 160 and has no CA.
                # In general, we can probably do r1-1 < len(ca_atom_list), etc.
                #
                continue
            else:
                ca1 = ca_atom_id_list[r1-1]
                ca2 = ca_atom_id_list[r2-1]
                ca_result[r1-1][r2-1] = all_atom_data[ca1-1][ca2-1]

            ma = max(all_vals)
            mi = min(all_vals)
            max_result[r1-1][r2-1] = ma
            min_result[r1-1][r2-1] = mi
            if abs(ma) > abs(mi):
                abs_result[r1-1][r2-1] = ma
            else:
                abs_result[r1-1][r2-1] = mi
    return {'abs':abs_result,
            'max':max_result,
            'min':min_result,
            'ca':  ca_result,
            }
def
write_max_min_ca_resi_versions(fname,ref_pdb_fname,non_ca_resis=[160,],overwrite=False):
    '''
    fname is the .dat file, a simple matrix file output by ptraj.
    ref_pdb_fname is used to get atom names and resi names, so that we can
                  properly select the max,min,abs,ca atoms from each resi.
    overwrite tells us whether or not to overwrite existing files.
    non_ca_resis is a list of resis that do not contain alpha carbons.
    '''
    fnames = {'max':'_resi_max'.join(os.path.splitext(fname))+'.bz2',
              'min':'_resi_min'.join(os.path.splitext(fname))+'.bz2',
              'abs':'_resi_abs'.join(os.path.splitext(fname))+'.bz2',
              'ca' :'_resi_ca'.join( os.path.splitext(fname))+'.bz2',
              }
    pre_existing_files = False
    for fn in fnames.values():
        if os.path.isfile(fn):
            if overwrite:
                print fn,"already exists, will overwrite"
            else:
                print fn,"already exists, will not overwrite"
                pre_existing_files = True
    if pre_existing_files and not overwrite:
        return
    atom_id_map = get_resi_to_atom_id_map(ref_pdb_fname)

master_residue_list,reference_coords,ca_atom_id_list,hydro_atom_id_list,nonhydro_atom_id_
list,nosp_atom_id_list,nonnosp_atom_id_list,mainchain_atom_id_list,sidechain_atom_id_list
,mainchain_nonhydro_atom_id_list,sidechain_hydro_atom_id_list,sidechain_nonhydro_atom_id_
list,skip_atom_id_list = setup_ca(ref_pdb_fname)
    sys.stdout.write('  reading in data\n')
    sys.stdout.flush()
    all_atom_data = read_data(fname)

    sys.stdout.write('  ready to calculate\n')
    sys.stdout.flush()
    data =
get_max_min_ca_resi_data(all_atom_data,atom_id_map,ca_atom_id_list,non_ca_resis=non_ca_re
sis)
    for d in data:
        if fnames[d].endswith('.bz2'):
            f = bz2.BZ2File(fnames[d],'w')
        else:
            f = file(fnames[d],'w')

        print "writing",fnames[d]
        io.write_array(f,data[d])
        f.close()

if __name__ == '__main__':
    #test_fname = '/Users/mglerner/tmp/Aligned/1rx1_trajectory_A_snap_0001_trans.pdb'
    #coords = parse_ca(test_fname,master_residue_list)
    if 0:
        #
        # Standard procedure for writing out the specific correlation and covariance
        # matricies that we want.
        #

        #for times in 'equil NS1 NS10 NS1-5 NS2-6 NS6-10 NS2-10'.split():
        #for times in 'NS2 NS3 NS4 NS5 NS6 NS7 NS8 NS9'.split():
        starts = range(500,9501,100)
        stops  = range(1500,10501,100)
        names  = [1 + i/10.0 for i in range(len(starts))]
        desired = [('%sps'%i,'%sps'%j,'NS%3.1f'%k) for (i,j,k) in
zip(starts,stops,names)]
        for times in [i[-1] for i in desired]:
            if 0:
                sys.stdout.write('doing %s 1RX1'%times)
                sys.stdout.flush()
                #dir = '/Users/mglerner/work/Dynamics-DHFR/MD_Files/1RX1/'
                dir = '/Users/mglerner/work/Dynamics-DHFR/BigDynamicsMoviePtrajFiles'

write_max_min_ca_resi_versions(os.path.join(dir,'1rx1_%s_all_atom_correlmat.dat'%times),
                                               '/Users/mglerner/work/Dynamics-
DHFR/MD_Files/1RX1/ChainAPDBFilesFromTrajectory/AlignedVs1RX1/1rx1_trajectory_A_snap_0001
_trans.pdb',
                                               )
            if 1:
                sys.stdout.write('doing %s 1RA1'%times)
                sys.stdout.flush()
                #dir = '/Users/mglerner/work/Dynamics-DHFR/MD_Files/1RA1/'
                dir = '/Users/mglerner/work/Dynamics-DHFR/BigDynamicsMoviePtrajFiles'

write_max_min_ca_resi_versions(os.path.join(dir,'1ra1_%s_all_atom_correlmat.dat'%times),
                                               '/Users/mglerner/work/Dynamics-
DHFR/MD_Files/1RA1/ChainAPDBFilesFromTrajectory/AlignedVs1RA1/1ra1_trajectory_A_snap_0001
_trans.pdb',
                                               )
    elif 1:
        #
        # Standard procedure for writing out the specific RMSD files that we want.
        #
        #for source_struct in '1RX1 1RA1'.split():
        for source_struct in '1RA1 1RX1'.split():
            print "SOURCE STRUCT",source_struct
            #for target_struct in '1RA1 1RX1 1RX2 1RX4 1RX5 1RX6'.split():
            for target_struct in '1RX1 1RA1'.split():
                if target_struct != source_struct:
                    continue
                print "TARGET STRUCT",target_struct
                data = {'ref_fname':'/Users/mglerner/work/Dynamics-
DHFR/CrystalStructureFiles/%sh_justatoms_A.pdb'%target_struct,
                        'target_pattern':'/Users/mglerner/work/Dynamics-
DHFR/MD_Files/%s/ChainAPDBFilesFromTrajectory/AlignedVs%s/%s_trajectory_A_snap_????_trans
.pdb'%(source_struct.upper(),target_struct,source_struct.lower()),

'rmsd_out_fname':'rmsd_ca_kellyaligned_%s_vs_%s.txt'%(source_struct.lower(),target_struct
),

'ileleu_out_fname':'ile50leu28_ca_kellyaligned_%s_vs_%s.txt'%(source_struct.lower(),targe
t_struct),
                        'ileleu_min_out_fname':
'ile50leu28_min_kellyaligned_%s_vs_%s.txt'%(source_struct.lower(),target_struct),
                        'ileleu_min_hydro_out_fname':
'ile50leu28_min_hydro_kellyaligned_%s_vs_%s.txt'%(source_struct.lower(),target_struct),
                        'ileleu_min_vdw_out_fname':
'ile50leu28_min_vdw_kellyaligned_%s_vs_%s.txt'%(source_struct.lower(),target_struct),
                        'ileleu_min_vdw_hydro_out_fname':
'ile50leu28_min_vdw_hydro_kellyaligned_%s_vs_%s.txt'%(source_struct.lower(),target_struct
),
                        'nap160phe31_out_fname':
'nap160phe31_kellyaligned_%s_vs_%s.txt'%(source_struct.lower(),target_struct),
                        'nap160phe31_min_out_fname':
'nap160phe31_min_kellyaligned_%s_vs_%s.txt'%(source_struct.lower(),target_struct),

'nap160phe31_min_hydro_out_fname':'nap160phe31_min_hydro_kellyaligned_%s_vs_%s.txt'%(sour
ce_struct.lower(),target_struct),

'ileasp_out_fname':'ile50asp27_ca_kellyaligned_%s_vs_%s.txt'%(source_struct.lower(),targe
t_struct),
                        'out_dir':'/Users/mglerner/work/Dynamics-DHFR/RMSD_Txt_Files/',
                        }
                write_rmsds_and_ile50leu28ca_dists(**data)
```

pypat/plotting.py

```
#!/usr/bin/env python
"""

TODO:

1) Get rid of unused functions
2) Possibly move the cmap lookup to tool_utils.

"""
from __future__ import division
from sentinel_map import SentinelMap,SentinelNorm
import glob
import matplotlib.numerix.ma as ma
import md_analysis_utils
import sys
import pylab,matplotlib,os
from matplotlib import mlab
import numpy as N
import scipy.io
import pprint
import bz2
import copy
from tool_utils import read_data
cdict = {'red':  (((-1.0+1)/2,32/255, 32/255),
                  ((-0.6+1)/2,32/255, 32/255),
                  ((-0.3+1)/2,32/255, 32/255),
                  ((-0.2+1)/2, 0/255,  0/255),
                  ((-0.1+1)/2, 0/255,  0/255),
                  (( 0.0+1)/2, 0/255,  0/255),
                  (( 0.1+1)/2, 0/255,  0/255),
                  (( 0.2+1)/2, 76/255, 76/255),
                  (( 0.3+1)/2,255/255,255/255),
                  (( 0.4+1)/2,255/255,255/255),
                  (( 0.5+1)/2,255/255,255/255),
                  (( 0.6+1)/2,242/255,242/255),
                  (( 0.7+1)/2,236/255,236/255),
                  (( 0.8+1)/2,236/255,236/255),
                  (( 0.9+1)/2,213/255,213/255),
                  (( 1.0+1)/2,108/255,108/255)),
         'green':(((-1.0+1)/2, 31/255, 31/255),
                  ((-0.6+1)/2, 31/255, 31/255),
                  ((-0.3+1)/2, 31/255, 31/255),
                  ((-0.2+1)/2, 86/255, 86/255),
                  ((-0.1+1)/2,131/255,131/255),
                  (( 0.0+1)/2,128/255,128/255),
                  (( 0.1+1)/2,137/255,137/255),
                  (( 0.2+1)/2,184/255,184/255),
                  (( 0.3+1)/2,244/255,244/255),
                  (( 0.4+1)/2,244/255,244/255),
                  (( 0.5+1)/2,172/255,172/255),
                  (( 0.6+1)/2, 52/255, 52/255),
                  (( 0.7+1)/2,  1/255,  1/255),
                  (( 0.8+1)/2,  0/255,  0/255),
                  (( 0.9+1)/2,  0/255,  0/255),
                  (( 1.0+1)/2, 24/255, 24/255)),
         'blue': (((-1.0+1)/2,109/255,109/255),
                  ((-0.6+1)/2,109/255,109/255),
                  ((-0.3+1)/2,109/255,109/255),
                  ((-0.2+1)/2,160/255,160/255),
                  ((-0.1+1)/2,202/255,202/255),
                  (( 0.0+1)/2,200/255,200/255),
                  (( 0.1+1)/2,144/255,144/255),
                  (( 0.2+1)/2, 45/255, 45/255),
                  (( 0.3+1)/2,  0/255,  0/255),
                  (( 0.4+1)/2,  0/255,  0/255),
                  (( 0.5+1)/2,  0/255,  0/255),
                  (( 0.6+1)/2,  0/255,  0/255),
                  (( 0.7+1)/2, 20/255, 20/255),
                  (( 0.8+1)/2, 22/255, 22/255),
                  (( 0.9+1)/2, 27/255, 27/255),
                  (( 1.0+1)/2, 33/255, 33/255)),
         }
scaled_by_half_cdict = {}
for color in cdict:
    bottom = cdict[color][0]
    top = cdict[color][-1]
    scaled_by_half_cdict[color] = [bottom,]
    for value in cdict[color]:
        scaled_by_half_cdict[color].append([0.25+value[0]/2,value[1],value[2]])
    scaled_by_half_cdict[color].append(top)
my_cmap = matplotlib.colors.LinearSegmentedColormap('my_colormap',cdict,256)
my_scaled_cmap =
matplotlib.colors.LinearSegmentedColormap('my_scaled_colormap',scaled_by_half_cdict,256)
##pprint.pprint( cdict['red'])
##pprint.pprint( scaled_by_half_cdict['red'])


sentinel1 = -10
sentinel2 = -20
rgb1 = (0.,0.,0.)
rgb2 = (1.,1.,1.)
#my_sentinelcmap = SentinelMap(my_cmap,sentinels={sentinel:rgb})
#my_sentinelnorm = SentinelNorm(ignore=[sentinel,],vmin=-1.0,vmax=1.0)
sentinel_maps_and_norms =
{'Normal':(SentinelMap(my_cmap,sentinels={sentinel1:rgb1,sentinel2:rgb2}),
                                     SentinelNorm(ignore=[sentinel1,sentinel2,],vmin=-
1.0,vmax=1.0)
                                     ),

'Scaled':(SentinelMap(my_scaled_cmap,sentinels={sentinel1:rgb1,sentinel2:rgb2}),
                                     SentinelNorm(ignore=[sentinel1,sentinel2,],vmin=-
1.0,vmax=1.0)
                                     ),
                           }


##                           'jet'
:(SentinelMap(pylab.cm.jet,sentinels={sentinel1:rgb1,sentinel2:rgb2}),
##                                     SentinelNorm(ignore=[sentinel1,sentinel2,],vmin=-
1.0,vmax=1.0)
##                                     ),
##                           }
for map in [m for m in pylab.cm.datad.keys() if not m.endswith("_r")]:
    sentinel_maps_and_norms[map] =
(SentinelMap(pylab.cm.get_cmap(map),sentinels={sentinel1:rgb1,sentinel2:rgb2}),
                                     SentinelNorm(ignore=[sentinel1,sentinel2,],vmin=-
1.0,vmax=1.0)
                                    )



def plotcontours_in_several_ways(fname='/Users/mglerner/work/Dynamics-
DHFR/MD_Files/1RX1/1rx1_byres_correlmat.dat'):
    data = scipy.io.read_array(file(fname))
    contourset = pylab.contourf(data)
    pylab.colorbar()
    pylab.show()
    pylab.pcolor(data)
    pylab.colorbar()
    pylab.show()
    pylab.pcolormesh(data)
    pylab.colorbar()
    pylab.show()
    del data

def plot_colormesh(fname,step=0.01,cmap=my_scaled_cmap):
    data = scipy.io.read_array(file(fname))
    #a = pylab.contourf(data,(-0.7,-0.2,0,0.2,0.6,1.0))
    a = pylab.contourf(data,pylab.arange(-1,1+step,step),cmap=cmap) # fine
    pylab.colorbar()
    pylab.show()
    del data
def plot_diff_colormesh(fn1,fn2):
    d1 = scipy.io.read_array(file(fn1))
    d2 = scipy.io.read_array(file(fn2))
    a = pylab.contourf(d1-d2,pylab.arange(-1,1,0.05),cmap=my_cmap) #coarse
    pylab.colorbar()
    pylab.show()


def plot_as_pcolor(fname):
    data = scipy.io.read_array(file(fname))
    a = pylab.pcolor(data,cmap=my_cmap)
    pylab.colorbar()
    pylab.show()
    del data




def plotrms(filenames):
    num_plots = len(filenames)
    pylab.figure(1)
    for i,fn in enumerate(filenames):
        pylab.subplot(num_plots,1,i+1)
        data = scipy.io.read_array(fn)
        pylab.plot([d[0] for d in data],[d[1] for d in data])
        pylab.title(fn[:-4])
        del data
    pylab.show()

def plotrms2(struct='1rx1'):
    md_data = scipy.io.read_array('%s_rms_vs_md.txt'%struct)
    ra1_data = scipy.io.read_array('%s_rms_vs_1RA1.txt'%struct)
    rx1_data = scipy.io.read_array('%s_rms_vs_1RX1.txt'%struct)
    rx2_data = scipy.io.read_array('%s_rms_vs_1RX2.txt'%struct)
    rx4_data = scipy.io.read_array('%s_rms_vs_1RX4.txt'%struct)
    rx5_data = scipy.io.read_array('%s_rms_vs_1RX5.txt'%struct)
    rx6_data = scipy.io.read_array('%s_rms_vs_1RX6.txt'%struct)
    #pylab.plot([d[0] for d in md_data],[d[1] for d in md_data],label='%s vs MD'%struct,)
    pylab.plot([d[0] for d in ra1_data],[d[1] for d in ra1_data],label='%s vs
1RA1'%struct,)
    pylab.plot([d[0] for d in rx1_data],[d[1] for d in rx1_data],label='%s vs
1RX1'%struct,)
    pylab.plot([d[0] for d in rx2_data],[d[1] for d in rx2_data],label='%s vs
1RX2'%struct,)
    #pylab.plot([d[0] for d in rx4_data],[d[1] for d in rx4_data],label='%s vs
1RX4'%struct,)
    pylab.plot([d[0] for d in rx5_data],[d[1] for d in rx5_data],label='%s vs
1RX5'%struct,)
    pylab.plot([d[0] for d in rx6_data],[d[1] for d in rx6_data],label='%s vs
1RX6'%struct,)
    pylab.legend()
    pylab.show()

def
plot_aligned_rms(struct='1rx1',structure_part='m20',in_dir='/Users/mglerner/work/Dynamics
-DHFR/RMSD_Txt_Files/',out_dir =
'',include_averages=False,prefix='rmsd_ca',title_mode='long',save_fig=True):
    if 'pymol' in prefix:
        file_title = 'pymol %s vs other structures (aligned, ca,
%s)'%(struct,structure_part)
    else:
        file_title = '%s vs other structures (aligned, ca, %s)'%(struct,structure_part)
    if include_averages: file_title = file_title + ' ' + 'ave'
    for s in '1RA1 1RX1 1RX2 1RX4 1RX5 1RX6'.split():
        fname =
os.path.join(in_dir,prefix+'_kellyaligned_%s_vs_%s_%s.txt'%(struct,s,structure_part))
        print "Trying to load",fname
        data = scipy.io.read_array(fname)
        if title_mode == 'long':
            title = '%s vs %s (aligned, ca, %s)'%(struct,s,structure_part)
        elif title_mode == 'short':
            title = 'vs. %s'%(s,)
        if include_averages:
            if len(data[0]) == 1:
                ave = sum(data)/len(data)
                title = title + ' Ave. %s'%ave
            elif len(data[0]) == 2:
                ave = sum([i[1] for i in data])/len([i[1] for i in data])
                title = title + ' Ave. %s'%ave
        if len(data.shape) == 1:
            pylab.plot([i*5/1000.0 for i in range(len(data))],data,label=title)
        elif len(data[0]) == 2:
            pylab.plot([i*5/1000.0 for i in range(len(data))],[i[1] for i in
data],label=title)
        else:
            sys.exit('what?')
        del data
    pylab.figlegend()
    a = pylab.axis()
    pylab.xticks([i/2.0 for i in range(24)])
    pylab.xticks([0,1,2,3,4,5,6,7,8,9,10],size='larger')
    pylab.yticks(size='larger')
    # the legend goes from 4.5 to 6 on a scale of 0-6, so about 25%
    pylab.axis((0,10.505,a[2],a[3]+0.25*(a[3]-a[2]))) # add one on the y axis so there's
room for the legend.
    pylab.grid(b=True)
    pylab.xlabel('Time (ns)',size='larger')
    pylab.rc('text',usetex=True)
    pylab.ylabel(r'RMSD ($\AA{}$)',size='larger')
    if save_fig:
        pylab.savefig(os.path.join('RMSDImages',out_dir,file_title+'.png'))
    pylab.show()
    pylab.clf()
```

```
def plot_ileleu(struct='1rx1'):
    for s in '1RA1 1RX1 1RX2 1RX4 1RX5 1RX6'.split():
        data =
scipy.io.read_array(file('ile50leu20_ca_kellyaligned_%s_vs_%s.txt'%(struct,s)))
        pylab.plot(range(len(data)),data,label='ILE LEU %s vs %s (aligned,
ca)'%(struct,s))
        del data
    pylab.legend()
    pylab.show()


def plotrms3():
    ra1_data = scipy.io.read_array('rms_vs_1RA1.txt')
    ra1_ca_data = scipy.io.read_array('rms_vs_1RA1_CA.txt')
    rx2_data = scipy.io.read_array('rms_vs_1RX2.txt')
    rx2_ca_data = scipy.io.read_array('rms_vs_1RX2_CA.txt')
    #pylab.plot([d[0] for d in ra1_data],[d[1] for d in ra1_data],label='vs 1RA1')
    pylab.plot([d[0] for d in ra1_ca_data],[d[1] for d in ra1_ca_data],label='vs 1RA1
CA')
    #pylab.plot([d[0] for d in rx2_data],[d[1] for d in rx2_data],label='vs 1RX2')
    pylab.plot([d[0] for d in rx2_ca_data],[d[1] for d in rx2_ca_data],label='vs 1RX2
CA')
    pylab.legend()
    pylab.show()

def OLDplot_ile_leu_dist():
    ra_ave_data = scipy.io.read_array('1ra1ile50leu28ave.dat')
    ra_ca_data = scipy.io.read_array('1ra1ile50leu28ca.dat')
    rx_ave_data = scipy.io.read_array('1rx1ile50leu28ave.dat')
    rx_ca_data = scipy.io.read_array('1rx1ile50leu28ca.dat')
    #pylab.plot([d[0] for d in ra_ave_data],[d[1] for d in ra_ave_data],label='1RA1
Average',)
    #pylab.plot([d[0] for d in ra_ca_data],[d[1] for d in ra_ca_data],label='1RA1 CA',)
    pylab.plot([d[0] for d in rx_ave_data],[d[1] for d in rx_ave_data],label='1RX1
Average',)
    pylab.plot([d[0] for d in rx_ca_data],[d[1] for d in rx_ca_data],label='1RX1 CA',)
    pylab.legend()
    pylab.show()

def
make_several_correl_plots(struct,times,cmap,detail_level,save_fig,out_dir='CorrelAndCovar
Images/',overwrite=True,
                              dat_dir='/Users/mglerner/work/Dynamics-
DHFR/BigDynamicsMoviePtrajFiles/',
                              ref_pdb_fname = '/Users/mglerner/work/Dynamics-
DHFR/MD_Files/1RX1/ChainAPDBFilesFromTrajectory/AlignedVs1RX1/1rx1_trajectory_A_snap_0001
_trans.pdb',
                              plot_types='ca avg max min abs straight mainheavy allheavy
sidechainhbond hbond'.split(),
                              mark_resis=[],
                              highlight=0.0,
                              highlight_mode='positive',
                              skip_resis=[],
                              ticks=True,
                              dpi=200,
                              title=None,
                              ):
    for plot_type in [i for i in 'abs max min avg ca'.split() if i in plot_types]:
        if plot_type == 'avg': fname =
os.path.join(dat_dir,'%s/%s_%s_byres_correlmat.dat'%(struct,struct,times))
        else:                  fname =
os.path.join(dat_dir,'%s/%s_%s_all_atom_correlmat_resi_%s.dat'%(struct,struct,times,plot_
type))
        data = read_data(fname)
        if data is None:
            continue

_make_one_correl_plot(data,struct,times,cmap,detail_level,plot_type,save_fig,dat_dir,out_
dir,overwrite,ref_pdb_fname,mark_resis,highlight,highlight_mode,skip_resis,ticks,dpi,titl
e)
        del data
    longer_plot_types = [i for i in 'straight mainheavy allheavy sidechainhbond
hbond'.split() if i in plot_types]
    if longer_plot_types:
        fname =
os.path.join(dat_dir,'%s/%s_%s_all_atom_correlmat.dat'%(struct,struct,times))
        data = read_data(fname)
        if data is None:
            return
        for plot_type in longer_plot_types:
            _data = copy.copy(data)

_make_one_correl_plot(data,struct,times,cmap,detail_level,plot_type,save_fig,dat_dir,out_
dir,overwrite,ref_pdb_fname,mark_resis,highlight,highlight_mode,skip_resis,ticks,dpi,titl
e)
            del _data
        del data


def
make_one_correl_plot(struct,times,cmap,detail_level,plot_type,save_fig,dat_dir,out_dir,ov
erwrite,ref_pdb_fname,mark_resis=[],highlight=0.0,highlight_mode='positive',skip_resis=[]
,ticks=True,dpi=200,title=None):
    if plot_type in 'straight mainheavy allheavy sidechainhbond hbond'.split(): fname =
os.path.join(dat_dir,'%s/%s_%s_all_atom_correlmat.dat'%(struct,struct,times))
    elif plot_type == 'avg':                                            fname =
os.path.join(dat_dir,'%s/%s_%s_byres_correlmat.dat'%(struct,struct,times))
    else:                                                               fname =
os.path.join(dat_dir,'%s/%s_%s_all_atom_correlmat_resi_%s.dat'%(struct,struct,times,plot_
type))
    data = read_data(fname)
    if data is None: return


_make_one_correl_plot(data,struct,times,cmap,detail_level,plot_type,save_fig,dat_dir,out_
dir,overwrite,ref_pdb_fname,mark_resis,highlight,highlight_mode,skip_resis,ticks,dpi,titl
e)

def
_make_one_correl_plot(data,struct,times,cmap,detail_level,plot_type,save_fig,dat_dir,out_
dir,overwrite,ref_pdb_fname,mark_resis,highlight,highlight_mode,skip_resis,ticks,dpi,titl
e=None):
    if title is None:
        title = struct+' '+times+' resi '+plot_type+' correl ('+detail_level+',
'+cmap+')'
    figname = os.path.join(out_dir,title+'.png')
    if os.path.exists(figname) and not overwrite:
        sys.stdout.write("Skipping "+figname+'\n')
    step = {'fine':0.01,'coarse':0.05}[detail_level]
    #
    # Now we split up the mark_resis. If it's just a list of resis,
    # we deal with it as such. If it's two lists, we'll use the
    # first list on the left axis and the second on the bottom one.
    #
    if mark_resis and type(mark_resis[0]) in (type(()),type([])) and len(mark_resis) ==
2:
        mark_left_resis,mark_bottom_resis = mark_resis
        mark_resis = sorted(mark_left_resis + mark_bottom_resis)
    else:
        mark_left_resis,mark_bottom_resis = mark_resis,mark_resis
    if plot_type in 'straight mainheavy allheavy sidechainhbond hbond'.split():
        #
        # These are the per-atom ones
        #

master_residue_list,reference_coords,ca_atom_id_list,hydro_atom_id_list,nonhydro_atom_id_
list,nosp_atom_id_list,nonnosp_atom_id_list,mainchain_atom_id_list,sidechain_atom_id_list
,mainchain_nonhydro_atom_id_list,sidechain_hydro_atom_id_list,sidechain_nonhydro_atom_id_
list,skip_atom_id_list =
md_analysis_utils.setup_ca(ref_pdb_fname,indexing=0,skip_resis=skip_resis)
        # We cannot to mark the before we yank stuff out.
        # If we do, we change the size of the array, and end up# taking the wrong
        # stuff out.  So, we set things up here and mark later.
        # That's also why left_bar must be padded later on.
        atom_id_map = md_analysis_utils.get_resi_to_atom_id_map(ref_pdb_fname,indexing=0)
        mark_left_atoms = []
        mark_bottom_atoms = []
        for resi in mark_left_resis:
            mark_left_atoms += atom_id_map[resi]
        for resi in mark_bottom_resis:
            mark_bottom_atoms += atom_id_map[resi]
        orig_size = data.shape[0]
        bottom_bar = N.array([[(sentinel1 if i in mark_bottom_atoms else sentinel2) for i
in range(orig_size)]])
        left_bar =   N.array([(sentinel1 if i in mark_left_atoms else sentinel2) for i in
range(orig_size)])
        left_bar.shape = (len(left_bar),1)

        if plot_type == 'mainheavy':
            data = scipy.take(data, mainchain_nonhydro_atom_id_list, axis = 0)
            left_bar = scipy.take(left_bar, mainchain_nonhydro_atom_id_list, axis = 0)
            data = scipy.take(data, mainchain_nonhydro_atom_id_list, axis = 1)
            bottom_bar = scipy.take(bottom_bar,mainchain_nonhydro_atom_id_list, axis = 1)
        elif plot_type == 'allheavy':
                totake = mainchain_nonhydro_atom_id_list+sidechain_nonhydro_atom_id_list
                totake.sort()
                data = scipy.take(data, totake, axis = 0)
                left_bar = scipy.take(left_bar, totake, axis = 0)
                data = scipy.take(data, totake, axis = 1)
                bottom_bar = scipy.take(bottom_bar,totake, axis = 1)


        elif plot_type == 'sidechainhbond':
            data = scipy.take(data,sidechain_hydro_atom_id_list, axis = 0)
            left_bar = scipy.take(left_bar,sidechain_hydro_atom_id_list, axis = 0)
            data = scipy.take(data,[i for i in nosp_atom_id_list if i not in
mainchain_nonhydro_atom_id_list], axis = 1)
            bottom_bar = scipy.take(bottom_bar,[i for i in nosp_atom_id_list if i not in
mainchain_nonhydro_atom_id_list], axis = 1)
            if 0 in data.shape:
                print "COULD NOT CALCULATE sidechainhbond"
                return
            #data = scipy.take(data,nosp_atom_id_list,    axis = 1)
        elif plot_type == 'hbond':
            # We could also consider a masked array.
            # We could use for i in atom_id_list:data[...,i] = sentinel
            # or a real mask.
            data       = scipy.take(data,       hydro_atom_id_list, axis = 0)
            left_bar   = scipy.take(left_bar,   hydro_atom_id_list, axis = 0)
            data       = scipy.take(data,       nosp_atom_id_list , axis = 1)
            bottom_bar = scipy.take(bottom_bar, nosp_atom_id_list , axis = 1)
            if 0 in data.shape:
                print "COULD NOT CALCULATE hbond"
                return
    else:
        #
        # These are the per-residue ones
        #
        data = scipy.take(data,[i for i in range(data.shape[0]) if i not in
skip_resis],axis=0)
        data = scipy.take(data,[i for i in range(data.shape[1]) if i not in
skip_resis],axis=1)
        orig_size = data.shape[0]
        bottom_bar = N.array([[(sentinel1 if i in mark_bottom_resis else sentinel2) for i
in range(orig_size)]])
        left_bar = [(sentinel1 if i in mark_left_resis else sentinel2) for i in
range(orig_size)]
        new_shape = (len(left_bar),1)
        left_bar = N.array(left_bar)
        left_bar.shape = new_shape

    if mark_resis:
        #
        # When we mark the matrix, we make the bottom bar the size of the original
        # data and we pad the left_bar with enough slack to take care of the bottom-
        # left corner.  After we're done marking, we will use the bottom bar and the
        # left bar for highlighting.  So, we'll need to go ahead and pad the bottom
        # bar at that point.
        #
        num_blocks = int(0.01*max(data.shape))+1
        for i in range(num_blocks): data = N.concatenate((bottom_bar,data))
        padding = N.array([sentinel2,]*num_blocks)
        padding.shape = (num_blocks,1)
        new_shape = (len(left_bar)+len(padding),1)
        left_bar = N.concatenate((padding,left_bar))
        for i in range(num_blocks): data = N.concatenate((left_bar,data),axis=1)
        padding.shape = 1,num_blocks
        bottom_bar = N.concatenate((padding,bottom_bar),axis=1)




    _cmap,_norm = sentinel_maps_and_norms[cmap]
    #plotter = pylab.pcolormesh
    plotter = pylab.imshow
    plotterargs = {pylab.imshow:{'interpolation':'nearest','origin':'lower'},
                   pylab.pcolormesh:{'shading':'flat'}}

    a = plotter(data,cmap=_cmap,norm=_norm,
                vmin=-1.0,vmax=1.0,**plotterargs[plotter])
    # For some reason, it draws the axes a little too long.
    if plot_type not in 'sidechainhbond hbond'.split():
        pylab.axis((0,data.shape[0],0,data.shape[1]))
    pylab.colorbar()
    pylab.title(title)
    if mark_resis and highlight:
        # Highlight the selected parts
        if highlight_mode in ('positive','negative'):
            mask = N.zeros(data.shape)
            for i in range(data.shape[0]):
                if left_bar[i] == sentinel1:
                    mask[i,] = True
            for j in range(data.shape[1]):
                if bottom_bar[0,j] == sentinel1:
                    mask[:,j] = True
            if highlight_mode == 'positive':
                data = N.ma.masked_where(N.logical_not(mask),data)
            elif highlight_mode == 'negative':
                data = N.ma.masked_where(mask,data)
            _cmap.set_bad('white',alpha=highlight)
            plotter(data,cmap=_cmap,norm=_norm,
                    vmin=-1.0,vmax=1.0,**plotterargs[plotter])

        elif highlight_mode == 'supernegative':
            last_val = left_bar[0]
            changes = []
            for i,val in enumerate(left_bar):
                if i == 0: continue
                if val != last_val:
                    changes.append(i)
                last_val = val
            for start,stop in zip(changes[:-1:2],changes[1::2]):
                facecolor='white'
                edgecolor='white'
                linewidth=0.0

pylab.axvspan(start,stop,0.01,1,fc=facecolor,ec=edgecolor,lw=linewidth,alpha=highlight)

pylab.axhspan(start,stop,0.01,1,fc=facecolor,ec=edgecolor,lw=linewidth,alpha=highlight)
            for i in changes:
                # I don't actually like the way this looks
                continue
                pylab.axhline(y=i,color='black')
                pylab.axvline(x=i,color='black')
    if ticks:
        from matplotlib.ticker import MultipleLocator, FormatStrFormatter, Formatter,
FixedLocator, LinearLocator, Locator

        class OffsetFormatStrFormatter(Formatter):
            def __init__(self,fmt,offset):
                self.fmt = fmt
                self.offset = offset
            def __call__(self,x,pos=None):
                return self.fmt % (x-self.offset)

        class ExplicitLinearLocator(Locator):
            def __init__(self,vmin,vmax,numticks):
                self.vmin = vmin
                self.vmax = vmax
                self.numticks=numticks
            def __call__(self):
                'Return the location of the ticks'
                ticklocs = mlab.linspace(self.vmin,self.vmax,self.numticks)
                return ticklocs

        offset = (num_blocks if mark_resis else 0)
        ymajorFormatter = OffsetFormatStrFormatter('%d',offset)
        xmajorFormatter = OffsetFormatStrFormatter('%d',offset)
        yminorLocator   = ExplicitLinearLocator(offset,data.shape[0],81)
        xminorLocator   = ExplicitLinearLocator(offset,data.shape[1],81)

        ax = pylab.gca()
        ax.yaxis.set_major_formatter(ymajorFormatter)
        ax.yaxis.set_minor_locator(yminorLocator)
        pylab.yticks(mlab.linspace(offset,data.shape[0],9))

        ax.xaxis.set_major_formatter(xmajorFormatter)
        ax.xaxis.set_minor_locator(xminorLocator)
        pylab.xticks(mlab.linspace(offset,data.shape[1],9))
    else:
        pylab.xticks([])
        pylab.yticks([])

    if save_fig:
        sys.stdout.write("Saving "+title+'\n')
        sys.stdout.flush()
        pylab.savefig(figname,dpi=dpi)
        pylab.clf()
    else:
        pylab.show()
    del data

def make_correl_plots_for_movie(structures=['1ra1','1rx1'],
                                all_times=['NS1.0,'],
                                cmaps=['Normal',],
                                detail_levels=['fine',],
                                plot_types='abs max min avg straight mainheavy allheavy
sidechainhbond hbond ca'.split(),
                                overwrite=True,
                                image_dir='BigMovieImages/',
                                dat_dir='/Users/mglerner/work/Dynamics-
DHFR/BigDynamicsMoviePtrajFiles/',
                                ref_pdb_fname = '/Users/mglerner/work/Dynamics-
DHFR/MD_Files/1RX1/ChainAPDBFilesFromTrajectory/AlignedVs1RX1/1rx1_trajectory_A_snap_0001
_trans.pdb',
                                mark_resis=[],
                                highlight=0.0,
                                highlight_mode='positive',
                                skip_resis=[],
                                ticks=True,
                                dpi=200,
                                title=None,
                                ):
    '''
    This will generate all of the .png images that you need to make your
    movies.  It will also spit out an html file called
    BigMovieImages<something>.html for you to look at.

    Empirical data: Setting width="86%" on all of the <img/> tags will make it so that
you can
    nicely fit the title and four rows of pictures on one page.
    '''
    for cmap in cmaps:
        for detail_level in detail_levels:
            fname = 'BigMovieImages'+cmap+detail_level+'.html'
            fout = file(fname,'w')
            fout.write('<html><head><title>Big Movie Images</title></head><body>')
            fout.write('<h1>'+cmap+' '+detail_level+'</h1><table>')
            made_plots_already = False
            for plot_type in plot_types:
                for times in all_times:
                    fout.write('<tr>')
                    for struct in structures:
                        title = struct+' '+times+' resi '+plot_type+' correl
('+detail_level+', '+cmap+')'
                        image_name = os.path.join(image_dir,title+'.png')
                        if os.path.exists(image_name) and not made_plots_already:
                            if overwrite:
                                print "Overwriting",title
                        if not made_plots_already:

make_several_correl_plots(struct,times,cmap,detail_level,save_fig=True,out_dir=image_dir,
overwrite=overwrite,
                                                      dat_dir=dat_dir,
                                                      ref_pdb_fname = ref_pdb_fname,
                                                      plot_types=plot_types,
                                                      mark_resis=mark_resis,
                                                      highlight=highlight,
                                                      highlight_mode=highlight_mode,
                                                      skip_resis=skip_resis,
                                                      ticks=ticks,
                                                      dpi=dpi,
                                                      title=title,
                                                      )

                        fout.write('<td><img src="'+image_name+'" width="86%" /></td>')
                    fout.write('</tr>')
                    fout.flush()
                fout.write('</tr>')
                fout.flush()

                made_plots_already=True
            fout.write('</table>')
            fout.write('</body></html>')
            fout.close()

def plot_rmsds_for_DHFR_paper():
    #start_time = 0.505 # in NS
    start_time = 0 # in NS
    stop_time  = 10.505 # in NS
    for time in 'xtal 505ps 1505ps 3005ps'.split():
        pylab.clf()
        data = {}
        data_dir = '/Users/mglerner/work/Dynamics-DHFR/MD_Files/RMSDAnalysis'
        data['rx_all'] =
scipy.io.read_array(os.path.join(data_dir,'1rx1_%s_all_atom_rmsds.txt'%time))
        data['rx_ca']  =
scipy.io.read_array(os.path.join(data_dir,'1rx1_%s_ca_rmsds.txt'%time))
        data['ra_all'] =
scipy.io.read_array(os.path.join(data_dir,'1ra1_%s_all_atom_rmsds.txt'%time))
        data['ra_ca']  =
scipy.io.read_array(os.path.join(data_dir,'1ra1_%s_ca_rmsds.txt'%time))
        #for struct in 'ra_all rx_all ra_ca rx_ca'.split():
        for struct in 'ra_all rx_all'.split():
        #for struct in 'ra_ca rx_ca'.split():
            labels = {#'ra_all':'1RA1 heavy atom RMSD vs %s (Avg. after = %s)',
                      'ra_all':'Open loop (Avg. = %s)',
                      'ra_ca' :'1RA1 ca RMSD vs %s                (Avg. after = %s)',
                      #'rx_all':'1RX1 heavy atom RMSD vs %s (Avg. after = %s)',
                      'rx_all':'Closed loop (Avg. = %s)',
                      'rx_ca' :'1RX1 ca RMSD vs %s                (Avg. after = %s)',
                      }
            this_data = [(i[0]*5/1000.0,i[1]) for i in data[struct] if ((i[0]*5/1000.0)
<= stop_time) and ((i[0]*5/1000.0) >= start_time)]
            data_after = [i[1] for i in this_data if (i[0] >=
{'xtal':0,'505ps':0.505,'1505ps':1.505,'3005ps':3.005}[time])]
```

```
            avg_after = sum(data_after)/len(data_after)
            print struct,avg_after,len(data_after)
            #pylab.plot([i[0]for i in this_data],[i[1] for i in
this_data],label=labels[struct]%(time,avg_after))
            pylab.plot([i[0]for i in this_data],[i[1] for i in
this_data],label=labels[struct]%(avg_after))
            #pylab.plot([i[0]for i in this_data],[avg_after for i in
this_data],color="black")
            f = file('/Users/mglerner/tmp/%s_MD_RMSDs_vs_%s.csv'%(struct,time),'w')
            f.write('Time,RMSD,Average %s RMSD after %s = %s\n'%(struct,time,avg_after))
            for i in this_data:
                f.write('%s,%s\n'%i)
            f.close()
        close_ticks = False
        if close_ticks:
            pylab.xticks([i/2.0 for i in range(24)])
            pylab.yticks([i/10.0 for i in range(30)])
            pylab.axis((start_time,stop_time,0,3))
            pylab.legend()
            pylab.grid(b=True)

pylab.savefig(os.path.join(data_dir,'1RA1and1RX1_MD_RMSDs_vs_%s_just_ca.png'%time))
        else:
            pylab.xticks([i/2.0 for i in range(24)])
            pylab.yticks([i/2.0 for i in range(30)])
            pylab.axis((start_time,stop_time,0,3))
            pylab.legend()
            pylab.grid(b=True)
            pylab.title('Closed- and Open-loop RMSDs vs. %s
structure'%{'xtal':'crystal'}.get(time,time))
            a = input()

pylab.savefig(os.path.join(data_dir,'1RA1and1RX1_MD_RMSDs_vs_%s_just_ca_fewer_ticks.png'%
time))


def plot_Sander_Data_for_DHFR_paper():
    pylab.clf()
    all_types = '1_4_EEL EELEC TEMP_K_ 1_4_NB EHBOND VDWAALS ANGLE EKCMT VIRIAL BOND
EKtot VOLUME CONSTRAINT EPtot DIHED Etot Density NSTEP EAMBER__non_constraint_
PRESS'.split()
    pos_energy_types = 'EKtot VDWAALS VIRIAL EKCMT 1_4_EEL ANGLE DIHED 1_4_NB
BOND'.split()
    neg_energy_types = 'Etot EPtot EELEC'.split()
    energy_types = pos_energy_types + neg_energy_types
    non_energies = 'VOLUME Density PRESS TEMP_K_'.split()
    for struct in '1rx1'.split():
    #for struct in '1ra1 1rx1'.split():
        start_dir = '/Users/mglerner/work/Dynamics-
DHFR/MD_Files/AllSanderOutFiles/Combined/%sfiles/'%struct
        if 0:
            # one-at-a-time
            for et in energy_types:
                data = scipy.io.read_array(os.path.join(start_dir,et+'.txt'))
                pylab.plot([(i[0]-500)/1000.0 for i in data],[i[1] for i in
data],color='gray',label=struct +'    '+et.replace('_',' '))
                last_part_of_data = [i[1] for i in data if ((i[0]-500)/1000.0) >= 1.5]
                avg = sum(last_part_of_data)/float(len(last_part_of_data))
                pylab.plot([(i[0]-500)/1000.0 for i in data],[avg for i in
data],color='black',label='Average after 1.5ns (%.3g)'%avg)
                print et,'\t %.3g'%avg
                pylab.xticks([i/2.0 for i in range(24)])
                pylab.grid(b=True)
                a = pylab.axis()
                pylab.axis((0,5.5,a[2],a[3]))
                pylab.legend(loc=0)

pylab.savefig(os.path.join('/Users/mglerner/work/DHFR/MD_Analysis',struct+'_'+et+'.png'))
                pylab.clf()
        if 1:
            # all at once
            for et in pos_energy_types:
                data = scipy.io.read_array(os.path.join(start_dir,et+'.txt'))
                avg = sum([i[1] for i in data])/float(len(data))
                pylab.plot([(i[0]-500)/1000.0 for i in data],[i[1] for i in
data],label=et+'(%.3g)'%avg)
                print et,'\t %.3g'%avg
                pylab.legend()
            pylab.xticks([i/2.0 for i in range(24)])
            pylab.grid(b=True)
            pylab.legend(loc=0)
            pylab.show()


def make_animated_gif():
    thumbPart = '-size 200x200 -geometry 200x200'
    for k in fileMap.keys():
        ps = getFilename(k,'.ps',imgDir)
        gif = getFilename(k,'.gif',imgDir)
        thumb = getFilename(k,'_thumb.gif',imgDir)
        os.system('/usr/bin/env convert -rotate 90 %s %s' % (ps,gif))
        os.system('/usr/bin/env convert -rotate 90 %s %s %s' % (thumbPart,
                                                                ps,
                                                                thumb))




if __name__ == '__main__':
    #plotrms(glob.glob('rms*.txt'))
    #plotrms2()
    #plot_ile_leu_dist()
    #plotcontours()

    #plotrms3()
    #plotcontours_in_several_ways()
    #plot_aligned_rms()
    #plot_ileleu()

    #plot_diff_colormesh('/Users/mglerner/work/Dynamics-
DHFR/MD_Files/1RX1/1rx1_byres_correlmat.dat','/Users/mglerner/work/Dynamics-
DHFR/MD_Files/1RA1/1ra1_byres_correlmat.dat')

    if 0:
        plot_rmsds_for_DHFR_paper()
    elif 0:
        #
        # Current correl/covar plotting mechanisms
        #
        #make_all_correl_plots()
        make_correl_plots_for_movie()
    elif 0:
        #
        # Current RMS plotting mechanisms for pymol-generated
        #
        pymol_generated_structure_parts = 'all_atom_substrate_binding_full
all_atom_substrate_binding_min n_substrate_binding_full n_substrate_binding_min
ca_substrate_binding_full ca_substrate_binding_min all_atom_substrate_binding_min_noR52
n_substrate_binding_min_noR52 ca_substrate_binding_min_noR52'.split()
        for (out_dir,structure_parts) in (('Standard',[i for i in
pymol_generated_structure_parts if 'R52' in i]),
                                          ):
            for structure_part in structure_parts:
                for struct in '1rx1 1ra1'.split():
                    for include_averages in (True,):#,False):
                        plot_aligned_rms(struct=struct,
                                         structure_part=structure_part,
                                         out_dir = out_dir,
                                         include_averages=include_averages,
                                         prefix='pymol_rmsd',
                                         )
    elif 0:
        #
        # Current RMS plotting mechanisms
        #
        #for structure_part in 'm20 FG CD GH all'.split():
```

```
        standard_parts = 'm20 all FG CD GH noloops subdomain1 subdomain2
substrate_binding substrate_binding2'.split()
        residue_chunks = '0-20 20-40 40-60 60-80 80-100 100-120 120-140 140-160'.split()
        anti_correlated_parts = 'm20 40-50 59-68 71-74 95-97'.split() + '116-124 142-
149'.split() # 116-124 is correlated with 40-50 and 59-68, 142-149 is correlated with 40-
50
        correlated_parts = '110-114 134-141 107-113 149-156 132-142 144-157'.split() +
'115-125 1-10 58-62 43-49 38-63 42-51'.split()
        for (out_dir,structure_parts) in (#('Standard',standard_parts),
                                          #('ResidueChunks',residue_chunks),
                                          #('AntiCorrelated',anti_correlated_parts),
                                          #('Correlated',correlated_parts),
                                          ('Standard',[i for i in standard_parts if
i.startswith('substrate_binding')]),
                                          ):
            for structure_part in structure_parts:
                for struct in '1ra1 1rx1'.split():
                    for include_averages in (True,):#,False):
                        plot_aligned_rms(struct=struct,
                                         structure_part=structure_part,
                                         out_dir = out_dir,
                                         include_averages=include_averages,
                                         prefix='rmsd_ca',
                                         )
```

pypat/sentinel_map.py

```
#!/usr/bin/env python2.4
#!/usr/bin/env python2.4
from __future__ import division
#from pylab import *
import pylab
from matplotlib.colors import Colormap, LinearSegmentedColormap, normalize
from matplotlib import numerix as nx
import matplotlib.numerix.ma as ma
from matplotlib.numerix import array, arange, alltrue
from types import IntType, FloatType, ListType

class SentinelMap(Colormap):
    def __init__(self, cmap=pylab.cm.jet, sentinels={}):
        # boilerplate stuff
        self.N = cmap.N
        self.name = 'SentinelMap'
        self.cmap = cmap
        self.sentinels = sentinels

        for rgb in sentinels.values():
            if len(rgb)!=3:
                raise ValueError('sentinel color must be RGB')
        self.is_gray = cmap.is_gray
        self.set_bad = cmap.set_bad


    def __call__(self, scaledImageData, alpha=1):
        # assumes the data is already normalized (ignoring sentinels)
        # clip to be on the safe side
        rgbaValues = self.cmap(nx.clip(scaledImageData, 0.,1.))
        for sentinel,rgb in self.sentinels.items():
            r,g,b = rgb
            if (scaledImageData==sentinel).max():
                rgbaValues[...,0] =  nx.where(scaledImageData==sentinel, r,
rgbaValues[...,0])
                rgbaValues[...,1] =  nx.where(scaledImageData==sentinel, g,
rgbaValues[...,1])
                rgbaValues[...,2] =  nx.where(scaledImageData==sentinel, b,
rgbaValues[...,2])
                rgbaValues[...,3] =  nx.where(scaledImageData==sentinel, alpha,
rgbaValues[...,3])
        return rgbaValues
```

```
class SentinelNorm(normalize):
    """
    Leave the sentinel unchanged
    """
    def __init__(self, ignore=[], vmin=None, vmax=None, clip = True):
        self.vmin=vmin
        self.vmax=vmax
        self.clip = clip

        if type(ignore) in [IntType, FloatType]:
            self.ignore = [ignore]
        else:
            self.ignore = list(ignore)
        self.ignore_mask=None

    def __call__(self, value, clip=None):

        if clip is None:
            clip = self.clip

        # ensure that we have a masked array val to work with
        if isinstance(value, (int, float)):
            vtype = 'scalar'
            val = ma.array([value])
        else:
            vtype = 'array'
            if ma.isMA(value):
                val = value
            else:
                val = ma.asarray(value)

        # create ignore_mask, val=sentinel1 | val= sentinel2..
        if self.ignore is not None:
            self.get_ignore_mask(val)

        # find min and max over points not masked by ignore_mask or by original mask of
val
        self.autoscale(val)

        # now do scaling
        vmin, vmax = self.vmin, self.vmax
        if vmin > vmax:
            if False in val.mask:
                raise ValueError("minvalue must be less than or equal to maxvalue")
            else:
                # array is completely masked. doesn't matter what values are for plot
                return 0.*value
        elif vmin==vmax:
            return 0.*value
        else:
            # scale points not masked by ignore_mask or by original mask of val
            scale = 1./(vmax-vmin)
            result = (val-vmin)*scale
            if clip:
                result = nx.clip(result,0.,1.)
            # set result over sentinel points to sentinel values
            if self.ignore is not None:
                result[self.ignore_mask]=val.data[self.ignore_mask]

        if vtype == 'scalar':
            result = result[0]
        return result

    def get_ignore_mask(self, A):
        if ma.isMA(A):
            A=A.data
        if self.ignore is not None:
            self.ignore_mask = False
            for ignore in self.ignore:
                self.ignore_mask |= A==ignore
    def autoscale(self, A):
        # self.scaled is method in base class Normalize [colors.py], is True if
self.vmin,vmax already defined
        if not self.scaled():
            if self.ignore is not None:
                if self.ignore_mask is None:
                    self.get_ignore_mask(A)
                A = ma.masked_where(self.ignore_mask,A)

            if self.vmin is None: self.vmin = A.min()
            if self.vmax is None: self.vmax = A.max()

    def inverse(self, value):
        if not self.scaled():
            raise ValueError("Not invertible until scaled")
        vmin, vmax = self.vmin, self.vmax

        if isinstance(value, (int, float)):
            return vmin + value * (vmax - vmin)
        else:
            val = ma.asarray(value)
            result = vmin + val * (vmax - vmin)
            if self.ignore is not None:
                if self.ignore_mask is None:
                    self.get_ignore_mask(value)
                result[self.ignore_mask]=val.data[self.ignore_mask]
            return result



cdict = {'red': ((0.0, 0.0, 0.0),
                 # This means that, at 0.5, Red starts at 1.0 and goes down
                 # to zero at 0.0.  It starts at 0.7 and goes up to 1.0 at 1.0.
                 (0.5, 1.0, 0.7),
                 (1.0, 0.0, 0.0)
                 ),
         'green': (#(0.0, 0.0, 0.0),
                   #(0.5, 1.0, 0.0),
                   #(1.0, 1.0, 0.0)
                   (0.0,0.0,0.0),
                   (0.5,0.0,0.0),
                   (1.0,0.0,0.0)
                   ),
         'blue': (#(0.0, 0.0, 0.0),
                  #(0.5, 1.0, 0.0),
                  #(1.0, 0.5, 0.0)
                   (0.0,0.0,0.0),
                   (1.0,0.0,0.0)
                   )
         }
cdict = {'red':  (((-1.0+1)/2,32/255, 32/255),
                  ((-0.6+1)/2,32/255, 32/255),
                  ((-0.3+1)/2,32/255, 32/255),
                  ((-0.2+1)/2,0,0),
                  ((-0.1+1)/2,0,0),
                  (( 0.0+1)/2,0,0),
                  (( 0.1+1)/2,0,0),
                  (( 0.2+1)/2, 76/255, 76/255),
                  (( 0.3+1)/2,255/255,255/255),
                  (( 0.4+1)/2,255/255,255/255),
                  (( 0.5+1)/2,255/255,255/255),
                  (( 0.6+1)/2,242/255,242/255),
                  (( 0.7+1)/2,236/255,236/255),
                  (( 0.8+1)/2,236/255,236/255),
                  (( 0.9+1)/2,213/255,213/255),
                  (( 1.0+1)/2,108/255,108/255)),
         'green':(((-1.0+1)/2, 31/255, 31/255),
                  ((-0.6+1)/2, 31/255, 31/255),
                  ((-0.3+1)/2, 31/255, 31/255),
                  ((-0.2+1)/2, 86/255, 86/255),
                  ((-0.1+1)/2,131/255,131/255),
                  (( 0.0+1)/2,128/255,128/255),
                  (( 0.1+1)/2,137/255,137/255),
                  (( 0.2+1)/2,184/255,184/255),
                  (( 0.3+1)/2,244/255,244/255),
                  (( 0.4+1)/2,244/255,244/255),
                  (( 0.5+1)/2,172/255,172/255),
                  (( 0.6+1)/2, 52/255, 52/255),
                  (( 0.7+1)/2,  1/255,  1/255),
                  (( 0.8+1)/2,  0/255,  0/255),
                  (( 0.9+1)/2,  0/255,  0/255),
                  (( 1.0+1)/2, 24/255, 24/255)),
         'blue': (((-1.0+1)/2,109/255,109/255),
                  ((-0.6+1)/2,109/255,109/255),
                  ((-0.3+1)/2,109/255,109/255),
                  ((-0.2+1)/2,160/255,160/255),
                  ((-0.1+1)/2,202/255,202/255),
                  (( 0.0+1)/2,200/255,200/255),
                  (( 0.1+1)/2,144/255,144/255),
                  (( 0.2+1)/2, 45/255, 45/255),
                  (( 0.3+1)/2,  0/255,  0/255),
                  (( 0.4+1)/2,  0/255,  0/255),
                  (( 0.5+1)/2,  0/255,  0/255),
                  (( 0.6+1)/2,  0/255,  0/255),
                  (( 0.7+1)/2, 20/255, 20/255),
                  (( 0.8+1)/2, 22/255, 22/255),
                  (( 0.9+1)/2, 27/255, 27/255),
                  (( 1.0+1)/2, 33/255, 33/255)),
         }
if __name__ == '__main__':
    # define the sentinels
    sentinel1 = 10
    sentinel2 = -10
    # define the colormap and norm
    rgb1 = (0.,0.,0.)
    rgb2 = (1.,1.,1.)


    #my_cmap =
SentinelLinearSegmentedColormap('my_colormap',cdict,256,sentinels={sentinel1:rgb1,sentine
l2:rgb2})
    my_cmap = LinearSegmentedColormap('my_colormap',cdict,256)
    norm = SentinelNorm(ignore=[sentinel1,sentinel2,sentinel_zero,sentinel_zero2])
    my_sentinelcmap = SentinelMap(my_cmap,sentinels={sentinel1:rgb1,
                                                     sentinel2:rgb2,
                                                     })


    #pcolor(rand(10,10),cmap=my_cmap)
    n = 100
    X = array(pylab.outerproduct(arange(-1,1,0.01),ones(100)))
    # replace some data with sentinels
    X[int(.1*n):int(.2*n), int(.5*n):int(.7*n)]  = sentinel1
    X[int(.6*n):int(.9*n), int(.2*n):int(.5*n)]  = sentinel2

    # make mask
    mask = nx.mlab.rand(n*2,n) >0.5
    print 'mask\n ',mask
    print mask.shape
    print X.shape


    # now mask X
    X= ma.masked_where(mask,X)

    #pylab.pcolormesh(X,cmap=my_cmap,shading='flat')
    pylab.pcolormesh(X,cmap=my_sentinelcmap,shading='flat',norm=norm)
    pylab.colorbar()
    pylab.show()
```

pypat/tool_utils.py

```
#!/usr/bin/env python
```

```
"""

Various utility code that gets used throughout the command-line tools.

Recent changes

 - Moved scipy and numpy imports into relevant functions so that
   other python builds (e.g. PyMOL) can easily use this.

"""

from __future__ import division
import os,sys,bz2
from copy import copy
from optparse import Option, OptionValueError

usage = """usage: %prog [options]

Please make sure that you have created the following directories:

  output-dir
  output-dir/structure-name
  output-dir/images
"""

############################
#
# New Optparse option types
#
############################

# Define a new option type, a comma-separated list of integers.
def check_zerobasedintlist(option,opt,value):
    """
    This automatically converts one-based indices to zero-based
    indices.  If you don't want that, make sure to convert back
    yourself!!

    It also knows about all of our lists of residues.
    """
    #
    # NOTE: if you want residues 9-24, give
    #       range(8,24) because we're zero-indexed.
    #
    standard_residue_lists = {
        # Loops
        'dhfr_fg':range(115,132),
        'dhfr_cd':range(63,71),
        'dhfr_m20':range(8,24),
        'dhfr_gh':range(141,150),
        'dhfr_subdomain1':range(37) + range(106,159),
        'dhfr_subdomain2':range(38,106),
        # Beta strands
        'dhfr_BA':[i-1 for i in range(  2,  5+1)],
        'dhfr_BB':[i-1 for i in range( 39, 43+1)], #
        'dhfr_BC':[i-1 for i in range( 58, 62+1)], #
        'dhfr_BD':[i-1 for i in range( 73, 75+1)], #
        'dhfr_BE':[i-1 for i in range( 91, 95+1)],
        'dhfr_BF':[i-1 for i in range(107,115+1)],
        'dhfr_BG':[i-1 for i in range(133,135+1)],
        'dhfr_BH':[i-1 for i in range(151,158+1)],
        # Alpha helices
        'dhfr_AB':[i-1 for i in range( 25, 35+1)],
        'dhfr_AC':[i-1 for i in range( 44, 50+1)], #
        'dhfr_AE':[i-1 for i in range( 78, 85+1)], #
        'dhfr_AF':[i-1 for i in range( 97,106+1)],

        # Networks
        'dhfr_AgarwalJPhysChemBNetwork':[i-1 for i in
(7,14,15,27,28,31,40,41,43,44,46,47,54,61,62,63,100,113,122)],
        'dhfr_WatneyHammes-SchifferJPhysChemBNetwork_shared':[i-1 for i in
(28,42,44,50,51,52,53,64,77,90)],
        'dhfr_WatneyHammes-SchifferJPhysChemBNetwork_ecolionly':[i-1 for i in
(19,20,45,47,61,62,63,67,68,72,74,98,99,102,108,129,149,155)],
        # Mutants
        'dhfr_rb_catcorr_mutants':[i-1 for i in (9,44,45,46,54,121,122)],
        'dhfr_rb_catnoncorr_mutants':[i-1 for i in (28,31,100,113)],
        'dhfr_rb_noncatnoncorr_mutants':[i-1 for i in (85,88,137,145,152,153,155)],
        'dhfr_rb_noncatcorr_mutants':[i-1 for i in (49,67,14,22)],
        # Other DHFR
        'dhfr_allosteric_site':[i-1 for i in        (      26,   29,30,   33,   111,
137,139,141,153,    155)],
        'dhfr_moe_allosteric_exposed':[i-1 for i in (      26,   29,30,   33,
137,139,141,153,    155)],
        'dhfr_moe_allosteric_tunnel': [i-1 for i in (5,6,7,   27,   30,31,
34,111,112,113,            153,154)],
        'dhfr_allosteric_everything': [i-1 for i in
(5,6,7,26,27,29,30,31,33,34,111,112,113,137,139,141,153,154,155)],

        'dhfr_allosteric_nonexposed': [i-1 for i in (5,6,7,   27,      31,   34,
112,113,                154)],

        'dhfr_mtx_contact':[i-1 for i in (5,6,27,31,32,52,57,94,100,160)],
        'dhfr_bulge_mutants':[i-1 for i in (137,153,155)],
        # BACE
        'bace_ticks':[i-1 for i in range(4,389,10)],
        }
    standard_residue_lists['dhfr_watney_network'] =
standard_residue_lists['dhfr_WatneyHammes-SchifferJPhysChemBNetwork_shared'] +
standard_residue_lists['dhfr_WatneyHammes-SchifferJPhysChemBNetwork_ecolionly'] + [159,]
    standard_residue_lists['dhfr_networks'] =
standard_residue_lists['dhfr_AgarwalJPhysChemBNetwork'] +
standard_residue_lists['dhfr_watney_network']
    standard_residue_lists['dhfr_loops'] = standard_residue_lists['dhfr_m20'] +
standard_residue_lists['dhfr_gh'] + standard_residue_lists['dhfr_fg'] +
standard_residue_lists['dhfr_cd']
    standard_residue_lists['dhfr_helices'] = []
    for i in 'AB AC AE AF'.split():
standard_residue_lists['dhfr_helices'].extend(standard_residue_lists['dhfr_'+i])
    standard_residue_lists['dhfr_rigid_d1'] = []
    for i in 'BB BC BD BE AC AE AF'.split():
standard_residue_lists['dhfr_rigid_d1'].extend(standard_residue_lists['dhfr_'+i])

    def parse_one_list(value):
        result = []
        for k in value.strip().split(','):
            if k in standard_residue_lists:
                result += standard_residue_lists[k]
            else:
                if '-' in k:
                    start,stop = k.split('-')
                    result.extend(range(int(start)-1,int(stop)))
                else:
                    result.append(int(k)-1)
        result = sorted(list(set(result)))
        print "Caring about",len(result),"residues",[i+1 for i in result]
        return result
    try:
        if ':' in value:
            left,right = value.split(':')
            return [parse_one_list(left),parse_one_list(right)]
        else:
            return parse_one_list(value)
    except ValueError:
        raise OptionValueError(
            "option %s is neither a known range of residues or a valid list of integers:
%s"%(opt,value))

# Define a new option type, a 1-based list of integers.  Just use
# the guts of the zero-based stuff and add one.

def check_onebasedintlist(option,opt,value):
    result = check_zerobasedintlist(option,opt,value)
    return [i+1 for i in result]
```

```
# Define a new option type, a comma-separated list of strings.
def check_strlist(option,opt,value):
    try:
        return [i for i in value.strip().split(',')]
    except ValueError:
        raise OptionValueError(
            "option %s: invalid list of strings: %s"%(opt,value))


class MyOption(Option):
    TYPES=Option.TYPES + ("zerobasedintlist","onebasedintlist","strlist",)
    TYPE_CHECKER=copy(Option.TYPE_CHECKER)
    TYPE_CHECKER["zerobasedintlist"] = check_zerobasedintlist
    TYPE_CHECKER["onebasedintlist"] = check_onebasedintlist
    TYPE_CHECKER["strlist"] = check_strlist

#######################################
#
# Adding options to the Optparse parser
#
#######################################

def add_window_options(parser):
    parser.add_option("--start",dest="start",default=500,type="int",
                      help="Time, in ps, to start the windows.  [default: %default]")
    parser.add_option("--stop",dest="stop",default=10500,type="int",
                      help="Time, in ps, to stop the windows.  [default: %default]")
    parser.add_option("--window-size",dest="windowsize",default=1000,type="int",
                      help="Length, in ps, of window size.  [default: %default]")
    parser.add_option("--window-spacing",dest="windowspacing",default=100,type="int",
                      help="Spacing between windows, in ps.  [default: %default]")

def add_standard_options(parser):
    parser.add_option("--structure-name",dest="structurename",
                      default="1rx1",
                      help="Name of your structure.  E.g. 1RX1 or 1SGZ. [default:
%default]")
    parser.add_option("--output-dir",dest="outputdir",
                      default="ptraj_files/",
                      help="Directory where we will put our results.  This should be the
same as the directory where we put our ptraj files before, and it should contain the .dat
files that ptraj outputs.  [default: %default]  The images will go to <outputdir>/images
and the html file will be in <outputdir>")

#######################################
#
# Parsing standard options
#
#######################################

def get_desired(options):
    starts = range(options.start,options.stop-options.windowsize+1,options.windowspacing)
    stops = range(options.start+options.windowsize,options.stop+1,options.windowspacing)
    names = [(starts[i] + options.windowsize/2.0)/1000.0 for i in range(len(starts))]
    # names is the centers of the windows.  start + windowsize/2 converted to NS.

    desired = [('%sps'%i,'%sps'%j,'NS%05.2f'%k) for (i,j,k) in zip(starts,stops,names)]
    return desired

#######################################
#
# Running external programs
#
#######################################

def run(prog,args,verbose=True):
    '''
    wrapper to handle spaces on windows.
    prog is the full path to the program.
    args is a string that we will split up for you.
        or a tuple.  or a list. your call.

    return value is (retval,prog_out)
    e.g.

    (retval,prog_out) = run("/bin/ls","-al /tmp/myusername")
    '''
    try:
        import subprocess,tempfile

        if type(args) == type(''):
            args = tuple(args.split())
        elif type(args) in (type([]),type(())):
            args = tuple(args)
        if os.name in 'nt dos'.split():
            # turns out not to be necessary with subprocess
            #prog = r'"%s"'%prog
            pass
        args = (prog,) + args
        output_file = tempfile.TemporaryFile(mode="w+")
        if verbose:
            print "Running",args
        retcode = 
subprocess.call(args,stdout=output_file.fileno(),stderr=subprocess.STDOUT)
        output_file.seek(0)
        prog_out = output_file.read()
        output_file.close() #windows doesn't do this automatically
        if verbose:
            print "Results were:"
            print "Return value:",retcode
            print "Output:"
            print prog_out
        return (retcode,prog_out)
    except ImportError:
        # probably python <= 2.4
        if type(args) != type(''):
            args = ' '.join(args)
        cmd = prog + ' ' + args
        if verbose:
            print "Running",cmd
        retcode = os.system(cmd)
        # cannot return prog_out via os.system
        if verbose:
            print "Results were:"
            print "Return value:",retcode
            print "Output:"
            print "\tcould not find subprocess module, so no output reported"
        return (retcode,'')

def read_data(fname):
    """
    Given a filename, try to read in data, paying attention to whatever format it's in.

    We'll try the following in order

     - .numpy
           We assume this is a square numpy array, made with array.tofile()
     - .dat
           A .dat file from ptraj.  This can be read in with scipy.io.read_array()
    """
    from scipy.io import read_array
    import numpy as N
    data = None
    sys.stdout.write("reading "+fname+" ")
    sys.stdout.flush()
    if os.path.isfile(fname+'.numpy'):
        sys.stdout.write("as numpy\n")
        sys.stdout.flush()
        f = file(fname+'.numpy')
        data = N.fromstring(f.read())
        f.close()
        one_side = int(N.sqrt(len(data)))
        if abs(len(data) - one_side*one_side) >= 0.1:
            message = "Only square matrices are supported. %s has len 
%s."%(fname,one_side)
```

```
            sys.stdout.write(message)
            return None
        data.shape = one_side,one_side
    elif os.path.isfile(fname+'.numpy.bz2'):
        sys.stdout.write("as numpy.bz2\n")
        sys.stdout.flush()
        bf = bz2.BZ2File(fname+'.numpy.bz2')
        data = N.fromstring(bf.read())
        bf.close()
        one_side = int(N.sqrt(len(data)))
        if abs(len(data) - one_side*one_side) >= 0.1:
            message = "Only square matrices are supported. %s has len
%s."%(fname,one_side)
            sys.stdout.write(message)
            return None
        data.shape = one_side,one_side

    elif os.path.isfile(fname):
        sys.stdout.write("as dat\n")
        sys.stdout.flush()
        data = read_array(file(fname))
    elif os.path.isfile(fname+'.bz2'):
        sys.stdout.write("as dat.bz2\n")
        sys.stdout.flush()
        data = read_array(bz2.BZ2File(fname+'.bz2'))
    else:
        sys.stdout.write("COULD NOT FIND (1)%s\n"%fname)
    return data
```

pypat/hbond/__init__.py

```
#!/usr/bin/env python
#blank
```

pypat/hbond/hbond_analysis.py

```
#!/usr/bin/env python

"""
Michael Lerner's hbond analysis

Right now, just handles pasting together ptraj output.
"""

import copy,pprint,os,sys

class Atom:
    def __init__(self,atom_number=None,resi=None,atom_name=None):
        """
        If we are involved in a solvent hydrogen bond, we expect
        that atom_number, resi and atom_name will all be 'solv'
        """
        self.atom_number = atom_number
        self.resi = resi
        self.atom_name = atom_name
    def __str__(self):
        if len(str(self.atom_number)) > 4:
            raise Exception ('Atom Number Too Long: %s'%self.atom_number)
        spaces = (4-len(str(self.atom_number))) * ' '
        number_str = spaces + '(%s)' % self.atom_number
        return "%3s %4s %s" %(self.resi,
                              self.atom_name,
                              number_str)
    __repr__ = __str__
    def __eq__(self,other):
        #print "Comparing atoms"
        #print "resi", (self.resi == other.resi)
        #print "name", (self.atom_name == other.atom_name)
        #print "number", (self.atom_number == other.atom_number)
        return (self.resi == other.resi) and (self.atom_name == other.atom_name) and
(self.atom_number == other.atom_number)
    def __ne__(self,other):
        return not (self == other)


class HBond:

    def init_from_str(self,s):
        '''
        str looks like what is returned by self._atom_str:
         41    O  (659) -->  94    H (1455) -  94    N (1454)
        '''

d_resi,d_name,d_number,arrow,ah_resi,ah_name,ah_number,dash,a_resi,a_name,a_number =
s.split()
        d_number,ah_number,a_number = [int(i.replace('(','').replace(')','')) for i in
d_number,ah_number,a_number]
        d_resi,a_resi,ah_resi = [int(i) for i in d_resi,a_resi,ah_resi]
        self.donor = Atom(d_number,d_resi,d_name)
        self.acceptorh = Atom(ah_number,ah_resi,ah_name)
        self.acceptor = Atom(a_number,a_resi,a_name)
    def __init__(self,line):
        '''
        Initialize ourself from a line that looks like this:
                DONOR         ACCEPTORH      ACCEPTOR
          atom# :res@atom   atom# :res@atom atom# :res@atom %occupied  distance
angle              lifetime maxocc
        |  2546 :160@OA23|  1018   :63@HG   1017   :63@OG  |  99.50  2.641 ( 0.10)  20.89
( 9.75)    100.0 ( 47.0)    147 |@@@@@@@@@@@@@*@@@@@|
        |  2545 :160@OA22|   705   :44@HH22  703   :44@NH2 |  98.51  2.756 (10.09)  17.97
(19.79)     99.0 (127.0)    126 |*@@@@@@@@@@*@@@@@@@|

        The numbers in parens are standard deviations.

        maxocc is the maximum number of consecutive frames where the H-bond exists.

        Here is a note from cheetham (http://amber.scripps.edu/Questions/mail/322.html)::

            The maxocc is the maximum number of consecutive frames that the
            interaction is found in the trajectory (i.e. 39 consecutive frames).

            The lifetime is the average time an interaction occurred...

            For example, assume that each space below represents 1ps and a star

            means it is occupied:

                    10        20         30        40        50
                 *****     *****     **********          *****|

            The occupancy would be 5 + 5 + 10 + 5 / 50 or 50%
            The maxocc would be 10
            The lifetime would be 5 + 5 + 10 + 5 / 4 = 6.25 ps (assuming 1 ps between
            frames; the time per frame can be specified on the hbond command line)

        So, we will need to adjust the lifetimes.  We have 5ps per frame, so our lifetime
        will need to be multiplied by 5.

        Adding things is not finished yet, but it works for
          - occ
          - distance
          - angle
          - graph
        '''

        # count tells us how many frames have been added together.
        self.count = 1
        if line is None:
            self.donor = Atom()
            self.acceptorh = Atom()
            self.acceptor = Atom()
            self.occ = self.dist = self.dist_stdev = self.angle = self.angle_stdev = 0.0
            self.lifetime = self.lifetime_stdev = 0.0
            self.maxocc = 0
            self.graph =  'XXXXXXXXXXXXXXXXXXX'
            self.graph =  '                   '
            self.graph =  '          '
            return

        line = line.strip()
        try:
            leading_junk,donor,acceptor,stats,graph,trailing_junk = line.split('|')
        except ValueError:
            print "Could not hbond",line
            raise
        self.donor = self._ptraj_hbond_chunk_to_atom(donor)
        self.acceptorh = self._ptraj_hbond_chunk_to_atom(' '.join(acceptor.split()[:2]))
        self.acceptor = self._ptraj_hbond_chunk_to_atom(' '.join(acceptor.split()[2:]))
        occ,dist = stats.split('(')[0].strip().split()
        dist_stdev = stats.split('(')[1].split(')')[0].strip()
        angle = stats.split(')')[1].split('(')[0].strip()
        angle_stdev = stats.split('(')[2].split(')')[0].strip()
        lifetime = stats.split('(')[-2].split()[-1].strip()
        lifetime_stdev = stats.split('(')[-1].split(')')[0].strip()
        maxocc = stats.split()[-1]
        self.occ,self.dist,self.dist_stdev,self.angle,self.angle_stdev = [float(i) for i
in occ,dist,dist_stdev,angle,angle_stdev]
        self.lifetime,self.lifetime_stdev = [float(i)*5 for i in lifetime,lifetime_stdev]
        self.maxocc = int(maxocc)*5
        #print
self.occ,self.dist,self.dist_stdev,self.angle,self.angle_stdev,self.lifetime,self.lifetim
e_stdev,self.maxocc
        #print graph,'->',graph.strip().strip('|')
        #self.graph = graph.strip().strip('|')
        self.graph = graph
    def _ptraj_hbond_chunk_to_atom(self,chunk):
        ''' chunk is something like "  2546 :160@OA23  " '''
        if chunk.strip() in ('solvent acceptor',
                             'solvent donor',
                             '',):
            if 'acceptor' in chunk:
                resn = 'acc'
            elif 'donor' in chunk:
                resn = 'don'
            else:
                resn = ''
            return Atom('solv','slv',resn)
        else:
            return Atom(int(chunk.split(':')[0].strip()),
                        int(chunk.split(':')[1].split('@')[0].strip()),
                        chunk.split(':')[1].split('@')[1].strip(),
                        )

    def __str__(self):
        return self._atom_str() + ' ' + self._occ_graph_str()
    def _atom_str(self):
        return "%s --> %s - %s"%(self.donor,
                                 self.acceptorh,
                                 self.acceptor,
                                 )
    def _occ_graph_str(self):
        return "occ:%6.2f(%2s) |%s|"%(self.occ,
                                      self.count,
                                      self.graph,)
    __repr__ = __str__
    def __add__(self,other):
        '''
        bleh.  i used to do things like
        result = copy.deepcopy(self)
        result.occ = (self.occ + other.occ)/2.0
        result.dist = (self.dist + other.dist)/2.0
        result.angle = (self.angle + other.angle)/2.0
        result.graph = self.graph + other.graph

        but, really, i need to add an attribute that
        tells me how long each hbond is for, so that when
        i add the fifth one in, it does not divide by two.

        for now, i will zero out all of the things just to
        make sure people understand that they are not to be
        believed.
        '''
        if type(self) != type(other):
            raise Exception('no can do, hombre')
        sep = '|'
        if other.acceptor.atom_number is None:
            result = copy.deepcopy(self)
            result.dist = result.angle = 0.0
            result.graph = self.graph + sep + other.graph
            result.count = self.count + other.count
        elif self.acceptor.atom_number is None or (self._atom_str() ==
other._atom_str()):
            result = copy.deepcopy(other)
            result.dist = result.angle = 0.0
            result.graph = self.graph + sep + other.graph
            result.count = self.count + other.count
        else:
            raise Exception('Can only add hbonds with the same donors and acceptors\n%s
!= %s'%(self._atom_str(),other._atom_str()))
        #
        # Add various parts now
        #
        result.occ = (self.count * self.occ + other.count * other.occ)/(self.count +
other.count)
        return result

def hbond_lines(lines):
    reading = False
    for line in lines:
        if line.strip() == '  atom# :res@atom   atom# :res@atom atom# :res@atom %occupied
distance       angle              lifetime maxocc'.strip():
            reading = True
        if not reading or line.strip().startswith('atom') or not line.replace('-
','').strip():
            continue
        yield line
def hbonds(f):
    return [HBond(line) for line in hbond_lines(f)]

def test_file_parsing():
    pprint.pprint(hbonds(file('/Users/mglerner/work/Dynamics-
DHFR/MD_Files/ptrajtestfiles/hbond_1rx1_ns2.out')))

def is_relevant(hbond,criteria):
    """
    Tells us if a hbond is relevant
    """
    if 'm20' in criteria:
        if ((9 <= hbond.donor.resi <= 24) or (9 <= hbond.acceptor.resi <= 24) or (9 <=
hbond.acceptorh.resi <= 24)):
            #print 'm20',hbond.donor.resi,hbond.acceptor.resi,hbond.acceptorh.resi
            return True
    if 'nap' in criteria:
        if (hbond.donor.resi == 160) or (hbond.acceptor.resi == 160) or
(hbond.acceptorh.resi == 160):
            #print "nap",hbond.donor.resi,hbond.acceptor.resi,hbond.acceptorh.resi
            return True
    if 'newpocket' in criteria:
        resis = 137,153,155,30,33,111
        if (hbond.donor.resi in resis) or (hbond.acceptor.resi in resis):
            return True
    if 'other' in criteria:
        return not (is_relevant(hbond,'m20') or is_relevant(hbond,'nap'))
    return False
    #return (9 <= hbond.donor.resi <= 24) or (9 <= hbond.acceptor.resi <= 24) or (9 <=
hbond.acceptorh.resi <= 24) or (hbond.donor.resi == 160) or (hbond.acceptor.resi == 160)
or (hbond.acceptorh.resi == 160)
```

```
def print_relevant_hbonds(fname,criterias):
    """
    Reads in one file and prints out the relevant stuff.
    This doesn't do any adding up of hbonds, etc.  It's
    mostly designed to work with my solvent hbonds.
    """
    hbonds = [HBond(line) for line in hbond_lines(file(fname))]

    for criteria in criterias:
        output = []
        for hbond in hbonds:
            if is_relevant(hbond,criteria):
                output.append((hbond.occ,hbond._atom_str()+' '+hbond._occ_graph_str()))
        output.sort()
        output.reverse()
        output = [o[1] for o in output]
        print '\n'.join(output)


def sum_hbonds(struct,criteria):
    print struct,criteria


    def add_hbonds(all_hbond_names,hbonds):
        """
        for every name in all_hbond_names that is not found in
        hbonds, add a blank hbond.
        """
        for hbond_name in all_hbond_names:
            found = False
            for hbond in hbonds:
                if hbond._atom_str() == hbond_name:
                    found = True
                    break
            if not found:
                hbond = HBond(None)
                hbond.init_from_str(hbond_name)
                hbonds.append(hbond)

    hbond_output_dir = '/Users/mglerner/work/Dynamics-DHFR/MD_Files/HBontOutputFiles'
    all_hbonds = {}
    all_hbond_names = {}
    for i in range(1,11):
        all_hbonds[i] = [HBond(line) for line in
hbond_lines(file(os.path.join(hbond_output_dir,'hbond_%s_NS%s.0.out'%(struct,i))))]
        for j in all_hbonds[i]:
            all_hbond_names[j._atom_str()] = None
    for i in all_hbonds:
        add_hbonds(all_hbond_names,all_hbonds[i])

    combined_hbonds = {}
    for hbond in all_hbonds[1]:
        combined_hbonds[hbond._atom_str()] = hbond
    for i in range(2,11):
        for hbond in all_hbonds[i]:
            combined_hbonds[hbond._atom_str()] = combined_hbonds[hbond._atom_str()] +
hbond
    output = []
    for k,v in combined_hbonds.iteritems():
        if is_relevant(v,criteria):
            #print k,v._occ_graph_str()
            output.append((v.occ,k+' '+v._occ_graph_str()))
    output.sort()
    output.reverse()
    output = [o[1] for o in output]
    print '\n'.join(output)



def test_hbond_constructors_and_overrides():
    hb1 = HBond('|  2546 :160@OA23|  1018   :63@HG   1017   :63@OG  |  99.50  2.641 (
0.10)  20.89 ( 9.75)    100.0 ( 47.0)    147 |@@@@@@@@@@@@@@*@@@@@|')
    print hb1
    hb2 = HBond('|  2545 :160@OA22|   705   :44@HH22  703   :44@NH2 |  98.51  2.756 
(10.09)  17.97 (19.79)     99.0 (127.0)    126 |*@@@@@@@@@@*@@@@@@@|')
    print hb2

    #print hb1+hb2

    hb3 = HBond('|  2546 :160@OA23|  1018   :63@HG   1017   :63@OG  |  20.50  2.641 (
0.10)  20.89 ( 9.75)    20.0 ( 17.0)     47 |------ooo--ooo@@@@@|')
    print hb3

    print hb1+hb3
```

pypat/hbond/hbond_definitions.py

```
#!/usr/bin/env python

############################
#
# A note to users:
# Feel free to edit this file to add your own
# hbond definitions. Python allows you to use single-
# and double-quotes pretty much interchangeably. In this
# file, however, you should use only double-quotes to quote
# things. Otherwise, it's too easy to mess things up because
# nucleic acids often use single-quotes as part of the atom names
# and we use those in our selections. In particular, I may run an
# automated procedure that adds in the "*" convention as well as the
# "'" convention, and those will break if you use "'" for anything else.
#
############################

from pymol import cmd

#
# RNA Defs from Mark Ditzler
#
def select_rna_acceptors():
    cmd.select("prot_acceptors","prot_acceptors or (resn RG and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG and name O6)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG and name N7)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG and name N3)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RG3 and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG3 and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG3 and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG3 and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG3 and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG3 and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG3 and name O6)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG3 and name N7)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG3 and name N3)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RG5 and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG5 and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG5 and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG5 and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG5 and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG5 and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG5 and name O6)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG5 and name N7)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RG5 and name N3)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RC and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC and name O2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC and name N3)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RC3 and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC3 and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC3 and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC3 and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC3 and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC3 and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC3 and name O2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC3 and name N3)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RC5 and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC5 and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC5 and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC5 and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC5 and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC5 and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC5 and name O2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RC5 and name N3)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RA and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA and name N7)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA and name N3)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA and name N1)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RA3 and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA3 and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA3 and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA3 and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA3 and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA3 and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA3 and name N7)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA3 and name N3)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA3 and name N1)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RA5 and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA5 and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA5 and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA5 and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA5 and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA5 and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA5 and name N7)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA5 and name N3)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RA5 and name N1)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RU and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU and name O2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU and name O4)")

    cmd.select("prot_acceptors","prot_acceptors or (resn RU3 and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU3 and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU3 and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU3 and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU3 and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU3 and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU3 and name O2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU3 and name O4)")
```

```
    cmd.select("prot_acceptors","prot_acceptors or (resn RU5 and name O2')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU5 and name O3')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU5 and name O4')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU5 and name O5')")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU5 and name O1P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU5 and name O2P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU5 and name O2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn RU5 and name O4)")
def select_rna_donors():
    """
    We only care about the hydrogen here, but we include the attached
    heavy atom in a comment for completeness.
    """
    #acceptor mask :RG@N1 :RG@H1
    cmd.select("prot_donors","prot_donors or (resn RG and name H1)")
    #acceptor mask :RG@N2 :RG@H21
    cmd.select("prot_donors","prot_donors or (resn RG and name H21)")
    #acceptor mask :RG@N2 :RG@H22
    cmd.select("prot_donors","prot_donors or (resn RG and name H22)")
    #acceptor mask :RG@O2' :RG@HO'2
    cmd.select("prot_donors","prot_donors or (resn RG and name HO'2)")

    #acceptor mask :RG3@N1 :RG3@H1
    cmd.select("prot_donors","prot_donors or (resn RG3 and name H1)")
    #acceptor mask :RG3@N2 :RG3@H21
    cmd.select("prot_donors","prot_donors or (resn RG3 and name H21)")
    #acceptor mask :RG3@N2 :RG3@H22
    cmd.select("prot_donors","prot_donors or (resn RG3 and name H22)")
    #acceptor mask :RG3@O2' :RG3@HO'2
    cmd.select("prot_donors","prot_donors or (resn RG3 and name HO'2)")
    #acceptor mask :RG3@O3' :RG3@H3T
    cmd.select("prot_donors","prot_donors or (resn RG3 and name H3T)")

    #acceptor mask :RG5@N1 :RG5@H1
    cmd.select("prot_donors","prot_donors or (resn RG5 and name H1)")
    #acceptor mask :RG5@N2 :RG5@H21
    cmd.select("prot_donors","prot_donors or (resn RG5 and name H21)")
    #acceptor mask :RG5@N2 :RG5@H22
    cmd.select("prot_donors","prot_donors or (resn RG5 and name H22)")
    #acceptor mask :RG5@O2' :RG5@HO'2
    cmd.select("prot_donors","prot_donors or (resn RG5 and name HO'2)")
    #acceptor mask :RG5@O5' :RG5@H5T
    cmd.select("prot_donors","prot_donors or (resn RG5 and name H5T)")

    #acceptor mask :RC@N4 :RC@H41
    cmd.select("prot_donors","prot_donors or (resn RC and name H41)")
    #acceptor mask :RC@N4 :RC@H42
    cmd.select("prot_donors","prot_donors or (resn RC and name H42)")
    #acceptor mask :RC@O2' :RC@HO'2
    cmd.select("prot_donors","prot_donors or (resn RC and name HO'2)")

    #acceptor mask :RC3@N4 :RC3@H41
    cmd.select("prot_donors","prot_donors or (resn RC3 and name H41)")
    #acceptor mask :RC3@N4 :RC3@H42
    cmd.select("prot_donors","prot_donors or (resn RC3 and name H42)")
    #acceptor mask :RC3@O2' :RC3@HO'2
    cmd.select("prot_donors","prot_donors or (resn RC3 and name HO'2)")
    #acceptor mask :RC3@O3' :RC3@H3T
    cmd.select("prot_donors","prot_donors or (resn RC3 and name H3T)")

    #acceptor mask :RC5@N4 :RC5@H41
    cmd.select("prot_donors","prot_donors or (resn RC5 and name H41)")
    #acceptor mask :RC5@N4 :RC5@H42
    cmd.select("prot_donors","prot_donors or (resn RC5 and name H42)")
    #acceptor mask :RC5@O2' :RC5@HO'2
    cmd.select("prot_donors","prot_donors or (resn RC5 and name HO'2)")
    #acceptor mask :RC5@O5' :RC5@H5T
    cmd.select("prot_donors","prot_donors or (resn RC5 and name H5T)")

    #acceptor mask :RA@N6 :RA@H61
    cmd.select("prot_donors","prot_donors or (resn RA and name H61)")
    #acceptor mask :RA@N6 :RA@H62
    cmd.select("prot_donors","prot_donors or (resn RA and name H62)")
    #acceptor mask :RA@O2' :RA@HO'2
    cmd.select("prot_donors","prot_donors or (resn RA and name HO'2)")

    #acceptor mask :RA3@N6 :RA3@H61
    cmd.select("prot_donors","prot_donors or (resn RA3 and name H61)")
    #acceptor mask :RA3@N6 :RA3@H62
    cmd.select("prot_donors","prot_donors or (resn RA3 and name H62)")
    #acceptor mask :RA3@O2' :RA3@HO'2
    cmd.select("prot_donors","prot_donors or (resn RA and name HO'2)")
    #acceptor mask :RA3@O3' :RA3@H3T
    cmd.select("prot_donors","prot_donors or (resn RA3 and name H3T)")

    #acceptor mask :RA5@N6 :RA5@H61
    cmd.select("prot_donors","prot_donors or (resn RA5 and name H61)")
    #acceptor mask :RA5@N6 :RA5@H62
    cmd.select("prot_donors","prot_donors or (resn RA5 and name H62)")
    #acceptor mask :RA5@O2' :RA5@HO'2
    cmd.select("prot_donors","prot_donors or (resn RA and name HO'2)")
    #acceptor mask :RA5@O5' :RA5@H5T
    cmd.select("prot_donors","prot_donors or (resn RA5 and name H5T)")

    #acceptor mask :RU@N3 :RU@H3
    cmd.select("prot_donors","prot_donors or (resn RU and name H3)")
    #acceptor mask :RU@O2' :RU@HO'2
    cmd.select("prot_donors","prot_donors or (resn RU and name HO'2)")

    #acceptor mask :RU3@N3 :RU3@H3
    cmd.select("prot_donors","prot_donors or (resn RU3 and name H3)")
    #acceptor mask :RU3@O2' :RU3@HO'2
    cmd.select("prot_donors","prot_donors or (resn RU3 and name HO'2)")
    #acceptor mask :RU3@O3' :RU3@H3T
    cmd.select("prot_donors","prot_donors or (resn RU3 and name H3T)")

    #acceptor mask :RU5@N3 :RU5@H3
    cmd.select("prot_donors","prot_donors or (resn RU5 and name H3)")
    #acceptor mask :RU5@O2' :RU5@HO'2
    cmd.select("prot_donors","prot_donors or (resn RU5 and name HO'2)")
    #acceptor mask :RU5@O5' :RU5@H5T
    cmd.select("prot_donors","prot_donors or (resn RU5 and name H5T)")

def select_standard_prot_donors_and_acceptors():
    """
    This is not standard because
    1) it knows about our terminal residues
    2) it includes our ligands
    """
    cmd.select("prot_acceptors","resn GLN and name OE1")
    cmd.select("prot_acceptors","prot_acceptors or (resn GLN and name OE1)")
    cmd.select("prot_acceptors","prot_acceptors or (resn GLN and name NE2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn ASN and name OD1)")
    cmd.select("prot_acceptors","prot_acceptors or (resn ASN and name ND2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn TYR and name OH)")
    cmd.select("prot_acceptors","prot_acceptors or (resn ASP and name OD1)")
    cmd.select("prot_acceptors","prot_acceptors or (resn ASP and name OD2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn GLU and name OE1)")
    cmd.select("prot_acceptors","prot_acceptors or (resn GLU and name OE2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn SER and name OG)")
    cmd.select("prot_acceptors","prot_acceptors or (resn THR and name OG1)")
    cmd.select("prot_acceptors","prot_acceptors or (resn HIS and name ND1)")
    cmd.select("prot_acceptors","prot_acceptors or (resn HIE and name ND1)")
    cmd.select("prot_acceptors","prot_acceptors or (resn HID and name NE2)")

    cmd.select("prot_donors","resn ASN and name HD21")
    cmd.select("prot_donors","prot_donors or (resn ASN and name 1HD2)")
    #acceptor mask  :ASN@ND2 :ASN@HD21
    cmd.select("prot_donors","prot_donors or (resn ASN and name HD21)")
    cmd.select("prot_donors","prot_donors or (resn ASN and name 1HD2)")
    #acceptor mask  :ASN@ND2 :ASN@HD22
    cmd.select("prot_donors","prot_donors or (resn ASN and name HD22)")
    cmd.select("prot_donors","prot_donors or (resn ASN and name 2HD2)")
    #acceptor mask  :TYR@OH  :TYR@HH
    cmd.select("prot_donors","prot_donors or (resn TYR and name HH)")
    #acceptor mask  :GLN@NE2 :GLN@HE21
    cmd.select("prot_donors","prot_donors or (resn GLN and name HE21)")
    cmd.select("prot_donors","prot_donors or (resn GLN and name 1HE2)")
    #acceptor mask  :GLN@NE2 :GLN@HE22
    cmd.select("prot_donors","prot_donors or (resn GLN and name HE22)")
    cmd.select("prot_donors","prot_donors or (resn GLN and name 2HE2)")
    #acceptor mask  :TRP@NE1 :TRP@HE1
    cmd.select("prot_donors","prot_donors or (resn TRP and name HE1)")
    #acceptor mask  :LYS@NZ  :LYS@HZ1
    cmd.select("prot_donors","prot_donors or (resn LYS and name HZ1)")
    #acceptor mask  :LYS@NZ  :LYS@HZ2
    cmd.select("prot_donors","prot_donors or (resn LYS and name HZ2)")
    #acceptor mask  :LYS@NZ  :LYS@HZ3
    cmd.select("prot_donors","prot_donors or (resn LYS and name HZ3)")
    #acceptor mask  :SER@OG  :SER@HG
    cmd.select("prot_donors","prot_donors or (resn SER and name HG)")
    #acceptor mask  :THR@OG1 :THR@HG1
    cmd.select("prot_donors","prot_donors or (resn THR and name HG1)")
    #acceptor mask  :ARG@NH2 :ARG@HH21
    cmd.select("prot_donors","prot_donors or (resn ARG and name HH21)")
    cmd.select("prot_donors","prot_donors or (resn ARG and name 1HH2)")
    #acceptor mask  :ARG@NH2 :ARG@HH22
    cmd.select("prot_donors","prot_donors or (resn ARG and name HH22)")
    cmd.select("prot_donors","prot_donors or (resn ARG and name 2HH2)")
    #acceptor mask  :ARG@NH1 :ARG@HH11
    cmd.select("prot_donors","prot_donors or (resn ARG and name HH11)")
    #acceptor mask  :ARG@NH1 :ARG@HH12
    cmd.select("prot_donors","prot_donors or (resn ARG and name HH12)")
    #acceptor mask  :ARG@NE  :ARG@HE
    cmd.select("prot_donors","prot_donors or (resn ARG and name HE)")
    #acceptor mask  :HIS@NE2 :HIS@HE2
    cmd.select("prot_donors","prot_donors or (resn HIS and name HE2)")
    #acceptor mask  :HIE@NE2 :HIE@HE2
    cmd.select("prot_donors","prot_donors or (resn HIE and name HE2)")
    #acceptor mask  :HID@ND1 :HID@HD1
    cmd.select("prot_donors","prot_donors or (resn HID and name HD1)")
    #acceptor mask  :HIP@ND1,NE2 :HIP@HE2,HD1
    cmd.select("prot_donors","prot_donors or (resn HIP and name HIP@HE2,HD1)")
    #-- Backbone donors and acceptors for this particular molecule
    #   N-H for prolines do not exist so are not in the mask.
    #


    #donor mask @O
    cmd.select("prot_acceptors","prot_acceptors or (name o and not resn WAT+HOH)")
    #   In our case, prolines are residues 21, 25, 39, 53, 55,
    #   66, 89, 105, 126, 130.  We would say 1-159, but we exclude
    #   prolines and terminii.
    #
    #acceptor mask :2-20,22-24,26-38,40-52,54,56-65,67-88,90-104,106-125,127-129,131-
158@N :1-158@H
    cmd.select("prot_donors","prot_donors or (name H and resi 2-158)")
    #Terminal residues have different atom names
    #donor mask @OXT
    cmd.select("prot_acceptors","prot_acceptors or name OXT")
    #acceptor mask :1@N :1@H1
    #acceptor mask :1@N :1@H2
    #acceptor mask :1@N :1@H3
    cmd.select("prot_donors","prot_donors or (resi 1 and name H1+H2+H3)")


def select_nap_donors_and_acceptors():
    """
    Ligand specific selections for NADPH (NAP)
    """
    #-- NADPH
    #acceptor mask :NAP@N6A  :NAP@H61
    cmd.select("prot_donors","prot_donors or (resn NAP and name H61)")
    #acceptor mask :NAP@N6A  :NAP@H62
    cmd.select("prot_donors","prot_donors or (resn NAP and name H62)")
    #acceptor mask :NAP@O'A3 :NAP@HOA3
    cmd.select("prot_donors","prot_donors or (resn NAP and name HOA3)")
    #acceptor mask :NAP@O'N3 :NAP@HON3
    cmd.select("prot_donors","prot_donors or (resn NAP and name HON3)")
    #acceptor mask :NAP@O'N2 :NAP@HON2
    cmd.select("prot_donors","prot_donors or (resn NAP and name HON2)")
    #acceptor mask :NAP@N7N  :NAP@H72
    cmd.select("prot_donors","prot_donors or (resn NAP and name H72)")
    #acceptor mask :NAP@N7N  :NAP@H71
    cmd.select("prot_donors","prot_donors or (resn NAP and name H71)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name N1A)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name N3A)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name N7A)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name OA23)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name OA22)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name OA24)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O'A2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O'A3)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O'A4)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O'A5)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name OPA1)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name OPA2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name OPN1)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O3P)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name OPN2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O'N5)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O'N4)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O'N3)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O'N2)")
    cmd.select("prot_acceptors","prot_acceptors or (resn NAP and name O7N)")


################################

def do_standard_selections():
    select_standard_prot_donors_and_acceptors()
    select_rna_donors()
    select_rna_acceptors()

cmd.extend("do_standard_selections",do_standard_selections)
```

pypat/hbond/pymol_hbond_analysis.py

```
#!/usr/bin/env python

"""

This version uses get_distance, etc. and takes around 10 hours per nanosecond.
That means that it'll take around 100 hours for my 10ns simulation.  If I split
it up onto 8 processors, that'll finish in about 12 or 13 hours.  So, about a
day to do the full hbond analysis.

TODO
----

Decide what to do for a default min_required_dwell_time.  The average
dwell time might be a more useful number if we exclude things that
are only there for e.g. 1 timestep.

Figure out what default for 'looseness' should be.

Perhaps build in a PyMOL interface that will highlight bridging waters
in a trajectory that you're currently viewing.

 - One way of doing this would be to just write out a new trajectory
   that included the appropriate waters in the appropriate states and
   then have the user load that in (or possibly load them in ourselves).

 - Another way would be to hide all of the waters that never make
   hydrogen bonds and then use dist mode=2.


REAL DOCUMENTATION
```

```
------------------

There are two main parts to this.

1) Use PyMOL to figure out what the bridging interactions are
   at each snapshot and spit them out to files.  That's typically
   done via a driver script like this::

       #!/usr/bin/env python
       import sys
       if sys.platform == 'darwin':
           print 'darwin'
           PYPAT_CODE_DIR = '/Users/mglerner/work/Dynamics-DHFR/'
       elif sys.platform == 'linux2':
           print 'linux2'
           PYPAT_CODE_DIR = '/users/mlerner/work/src/Dynamics-DHFR/'

       sys.path.append(PYPAT_CODE_DIR)
       from pypat.hbond import pymol_hbond_analysis
       r = range(2501,3001,500)
       starts_and_stops = zip(r[:-1],r[1:])
       for (start,stop) in starts_and_stops:
           print
           print 'DOING',start,stop
           print
           pymol_hbond_analysis.find_bridging_waters_in_trajectory('1ra1',start,stop)


2) Parse those results, invert them so they're in terms of bridging
   interactions rather than individual waters, spit out the results.
   That's typically done like this (and is done in the __main__
   loop here)::

    import glob
    d = '/Users/mglerner/work/Dynamics-DHFR/MD_Files/BridgingWaterOutput/'
    fnames = glob.glob(os.path.join(d,'1rx1_hbonds_*_*.txt'))
    a = get_hbond_trajectories('1rx1',
                               timestep=5,
                               fnames=fnames,
                               combination_method='loose',
                               min_required_dwell_time=3,
                               looseness=2,
                               dist_cutoff=3.5,
                               )
    including_resis =  137,153,155,30,33,111
    including_resis = None
    print a.get_trajectory_string(minocc=0.20,
                                  numchunks=50,
                                  including_resis=including_resis,
                                  )



"""
from __future__ import division

try:
    enumerate
except NameError:
    def enumerate(thing):
        result = []
        idx = 0
        for t in thing:
            result.append((idx,t))
            idx += 1
        return result


try:
    sum
except NameError:
    def sum(thing):
        result = 0
        for t in thing:
            result += t
        return result



import sys,os


class TrajectoryFormatter:
    '''
    From ptraj action.c
          if ( m > 0.95 * pt )
            fprintf(fpout, "@");
          else if ( m > 0.80 * pt )
            fprintf(fpout, "*");
          else if ( m > 0.60 * pt )
            fprintf(fpout, "x");
          else if ( m > 0.40 * pt )
            fprintf(fpout, "o");
          else if ( m > 0.20 * pt )
            fprintf(fpout, "-");
          else if ( m > 0.05 * pt )
            fprintf(fpout, ".");
          else
            fprintf(fpout, " ");
    '''
    def __init__(self):
        pass
    def equal_split(self,L,N):
        """
        Split L into N parts, returning a list containing those parts.  The last part may
be smaller than the others.
        """
        part = int((len(L) + N - 1)/ N)
        _L = []
        for i in range(0,N):
            _L.append( L[part*i : part*i + part] )
        return [i for i in _L if i]

    def format(self,mintime,maxtime,timestep,numchunks,trajectory):
        time_chunks =
self.equal_split(range(mintime,maxtime+timestep,timestep),numchunks)
        result = ''
        for r in time_chunks:
            occ = 0.
            for t in r:
                if trajectory[t]: occ += 1
            occ = occ/len(r)

            if occ > 0.95:
                result += "@"
            elif occ > 0.80:
                result += "*"
            elif occ > 0.60:
                result += "x"
            elif occ > 0.40:
                result += "o"
            elif occ > 0.20:
                result += "-"
            elif occ > 0.05:
                result += "."
            else:
                result += " "
        return result
    def get_description_string(self):

        return """
Trajectory formatting codes:
1.0-0.95 0.95-0.80 0.80-0.60 0.60-0.40 0.40-0.20 0.20-0.05 0.05-0.0
@@@@@@@@ ********* xxxxxxxxx oooooooo --------- .........
"""
```

```
default_trajectory_formatter = TrajectoryFormatter()



class BridgingWaterTrajectoryAnalyzer:
    """

    """
    def __init__(self,structure,timestep=5):
        """

        You are expected to run code like this:

        a = BridgingWaterTrajectoryAnalyzer(structure,timestep)
        for fname in fnames:
            a.read_in_file(fname)
        a.finalize()
        return a

        If you don't call finalize, we will still be able to output some
        statistics, but our trajectories will just be lists of
        SingleSnapshotBridgingWater instances, so you'll miss out on a
        lot of the output power.

        Parameters
        ----------

        structure: protein structure, e.g. 1RX1

        timestep: the number of picoseconds each frame represents.
        """
        self.struct = structure
        self.timestep = timestep
        self.mintime = 1000000000
        self.maxtime = 0
        #
        # self.trajectories is a dictionary that maps SingleSnapshotBridgingWater objects
        # to a series of times.  We'll set it up as a defaultdict so that you can ask
        # for trajectories[SSBW][time] and it'll return True or False.  True if we've
        # explicitly added things and false if we haven't.
        #
        """
        I think this might actually work better if self.trajectories[x] returned
        a trajectory, where x is the hash of a SSBW.  we then call
        self.trajectories[x][time] = x.wat_resi.

        so, that means that a trajectory has to have a __getitem__ and __setitem__.
        """
        self.trajectories = {}
    def __str__(self):
        results = []
        for bwt in self.trajectories:
            result = '%-103s'%bwt + ': ' + bwt.get_trajectory_string()

            results.append(result)
        return '\n'.join(results)

    def get_trajectory_string(self,minocc,numchunks,including_resis,sort_by):
        """
        Parameters
        ----------

        minocc: the minimum occupancy time required to return a string

        numchunks: the number of chunks into which we will divide the output.
                   If this is 10, you'll get 10 points in the output string, etc.
                   If this is None, you'll get the whole thing.

        including_resis: we will print out all trajectories that have at least one
                         leg involving at least one of these residues.

        sort_by: The attribute by which the trajectories will be sorted.  Usual
                 attributes are occ and average_dwell_time.  This should be a
                 string, as we will use it in a getattr() call.

        """
        global default_trajectory_formatter
        bwts = [(getattr(bwt,sort_by),bwt) for bwt in self.trajectories]
        print "BWTA getting %s trajectory strings with minocc %s, numchunks %s,
including_resis %s"%(len(bwts),

minocc,

numchunks,

including_resis,

)
        print default_trajectory_formatter.get_description_string()
        bwts.sort()
        bwts.reverse()
        bwts = [thing[1] for thing in bwts]
        return '\n'.join(['%-103s :: '%bwt + bwt.get_trajectory_string(numchunks) for bwt
in bwts if bwt.occ >= minocc and bwt.contains_resi(including_resis)])

    def
as_edges(self,bridging_water_info,edge_length=2,is_valid=None,dist_cutoff=9999999.0,angle
_cutoff=0.0):
        """
        Returns a bridging_water_info object as one SingleSnapshotBridgingWater per edge.

        e.g. ['7464', ('53', 'PRO', 'O', 'donatingto',2.2,'angle',160.0),
                      ('52', 'ARG', 'HE', 'acceptingfrom',2.4,'angle',165.0),
                      ('52', 'ARG', 'HH22', 'acceptingfrom',3.5,'angle',164.0)]

             will be returned as
               [SSBW(7464,('53', 'PRO', 'O',  'donatingto',   2.2,'angle',160.0),('52',
'ARG', 'HE',   'acceptingfrom',2.4,'angle',165.0),),
                SSBW(7464,('53', 'PRO', 'O',  'donatingto',   2.2,'angle',160.0),('52',
'ARG', 'HH22', 'acceptingfrom',3.5,'angle',164.0)),
                SSBW(7464,('52', 'ARG', 'HE', 'acceptingfrom',2.4,'angle',165.0),('52',
'ARG', 'HH22', 'acceptingfrom',3.5,'angle',164.0))]

        This is all subject to the is_valid function.  For instance, an is_valid function
        that says that you can't be bridging between two hydrogens on the same residue
would
        reject the last entry.

        Parameters
        ----------

        is_valid: a function to tell us if a bridging water is valid.  For example, we
might
                  want to reject bridges between two hydrogens on the same residue.  If
this
                  is None, we will use our default is_valid function, defined below.

        dist_cutoff: if you're using our is_valid function, it will reject edges where
either
                     bond is longer than dist_cutoff.

        angle_cutoff: if you're using our is_valid function, it will reject edges where
either
                      angle is less than angle_cutoff

        edge_length: length of the edges that we care about.  Default is two. I suppose
it's
                     not really an 'edge' if it's bigger than two, but you know what I
mean.
                     It's the list of things that are connected to this water.
        """
        if edge_length != 2:
            raise NotImplementedError('Only edge lengths of two are supported')
        results = []
        if is_valid is None:
            """
            Anything with None in it is invalid, and is just a default arg to the
            Hbond constructor

            We do not allow bridges between two hydrogens that are part of the same
            residue.
            """
            def is_valid(atom1,atom2,dist_cutoff=dist_cutoff,angle_cutoff=angle_cutoff):
                """

                an atom looks like ('111', 'TYR', 'O', 'donatingto',3.2, 'angle',160.0)
                """
                #
                # Reject default args (None)
                #
                if None in atom1:
                    return False
                if None in atom2:
                    return False
                #
                # Reject bridges between two hydros that are part of the same residue
                #
                if atom1[0] == atom2[0]: # same residue
                    if 0: print "same residue hydros",atom1,atom2
                    if atom1[2].startswith('H') and atom2[2].startswith('H'):
                        #print "rejecting",atom1,atom2
                        return False
                #
                # Bridges must be between two different residues
                #
                if atom1[0] == atom2[0]:
                    if 0: print "same residue",atom1,atom2
                    return False

                #
                # All edges must be <= the distance cutoff
                #
                if (atom1[4] > dist_cutoff) or (atom2[4] > dist_cutoff):
                    if 0:
                        print "bad dist",atom1,atom2
                    return False
                #
                # All angles must be >= the angle cutoff
                #
                if (atom1[6] < angle_cutoff) or (atom2[6] < angle_cutoff):
                    if 0:
                        print "bad angle",atom1,atom2
                    return False
                return True
        wat_resi,atoms = bridging_water_info[0],bridging_water_info[1:]
        for i,atom1 in enumerate(atoms):
            for atom2 in atoms[i+1:]:
                if is_valid(atom1,atom2):
                    results.append(SingleSnapshotBridgingWater(wat_resi,atom1,atom2))
        return results

    def finalize(self,combination_method,min_required_dwell_time,looseness):
        """
        Turn our trajectories into proper trajectories so that we can print out nicely.

        It also calculates a lot of statistics.

        Please look at BridgingWaterTrajectory.finalize() for more documentation.
        """
        sys.stdout.write("Now finalizing %s trajectories\n"%len(self.trajectories))
        sys.stdout.flush()
        for bwt in self.trajectories:
            bwt.mintime = self.mintime
            bwt.maxtime = self.maxtime
            bwt.timestep = self.timestep
            bwt.finalize(combination_method=combination_method,
                         min_required_dwell_time=min_required_dwell_time,
                         looseness=looseness)
    def read_in_file(self,fname,times=None,dist_cutoff=999999.,angle_cutoff=0):
        """

        Parameters
        ----------

        fname: The name of the file we should read in.
               The contents of this file will be eval()'d, and we expect
               to be able to turn the result of that into a dictionary.
               The keys in that dictionary are PyMOL object names and the
               values are lists of bridging waters found in those objects.

        times: A dictionary mapping the PyMOL object names to times.
               If times is None, we will determine the times from the
               filename as follows:

               1rx1_hbonds_635_640.txt means that times is range(635,640)
               with the special exception that, due to how PyMOL treats
               trajectories, there's never a time == 0, so 0_x will be
               range(1,x).

               In that case, we'll map the times to the lexigraphical
               ordering of the keys in fname.

               Times will be in picoseconds, and will be multiplied by
               self.timestep in order to convert from snapshot number
               to picoseconds.

               This function takes care of some bookkeeping as well by
               ensuring that self.mintime and self.maxtime are correct.
               Any other functions that touch self.trajectories should
               make sure to do this!

        dist_cutoff: if you're using our is_valid function, it will
                     reject edges where either bond is longer than
                     dist_cutoff.

        angle_cutoff: if you're using our is_valid function, it will
                      reject edges where either angle is smaller than
                      angle_cutoff.

        """
        f = file(fname)
        try:
            x = eval(f.read())
        except SyntaxError:
            print "Could not read file %s .. possibly still being written to?"%fname
            return
        f.close()
        if times is None:
            just_fname = os.path.splitext(os.path.split(fname)[-1])[0]
            parts = just_fname.split('_')
            start,stop = int(parts[-2]),int(parts[-1])
            if start == 0:
                start = 1
            times = {}
            for (o,t) in zip(sorted(x.keys()),range(start,stop)):
                times[o] = t*self.timestep
            if 0:
                print "Time mapping",times
        for obj_name,bridging_water_infos in x.iteritems():
            t = times[obj_name]
            if t < self.mintime:
                self.mintime = t
            if t > self.maxtime:
                self.maxtime = t
            for bridging_water_info in bridging_water_infos:
                wat_resi = int(bridging_water_info[0])
                for bw in
self.as_edges(bridging_water_info,dist_cutoff=dist_cutoff,angle_cutoff=angle_cutoff):
                    self.add_to_trajectory(bw,wat_resi,t)
    def add_to_trajectory(self,bw,wat_resi,t):
        if bw not in self.trajectories:
            il = bw.get_nonempty_interaction_list(include_distances=True)
            if 0:
                print il
            if len(il) != 2:
                raise NotImplementedError('We only support bridges with two edges at this
time %s'%il)
            if 0:
                print il[0]
            resi1,resn1,atomname1,interaction1,dist1 = il[0]
            resi2,resn2,atomname2,interaction2,dist2 = il[1]
            #
            # TODO: FIXME: we should pay more attention to the distance.
            # We should store it with the time in self.trajectories.
            #
            if 0:
                print "interaction1",interaction1
                print "interaction2",interaction2
            bwt = BridgingWaterTrajectory((resi1,resn1,atomname1,interaction1,dist1,),
                                          (resi2,resn2,atomname2,interaction2,dist2,),
                                          )

            self.trajectories[bwt] = bwt
            if 0:
                print
"added",(resi1,resn1,atomname1,interaction1,dist1,),(resi2,resn2,atomname2,interaction2,d
ist2,)

        # It's worth knowing that I've overloaded __setitem__ so that it will
        # actually just append wat_resi to the list of times.x
        try:
            self.trajectories[bw][t] = wat_resi
        except KeyError:
            print "trouble adding",bw
            raise


class BridgingWaterTrajectory:
    """
        I think this might actually work better if self.trajectories[x] returned
        a trajectory, where x is the hash of a SSBW.  we then call
        self.trajectories[x][time] = x.wat_resi.

        so, that means that a trajectory has to have a __getitem__ and __setitem__.

    """

    def __init__(self,
                 (resi1,resn1,atomname1,interaction1,dist1),
                 (resi2,resn2,atomname2,interaction2,dist2),
                 ):
        #
        # If we every use more trajectory formatters,
        # we'll make it an argument to __init__.
        #
        global default_trajectory_formatter
        from collections import defaultdict

        self.resi1 = int(resi1)
        self.resn1 = resn1
        self.atomname1 = atomname1
        self.interaction1 = interaction1
        self.resi2 = int(resi2)
        self.resn2 = resn2
        self.atomname2 = atomname2
        self.interaction2 = interaction2

        self._times = defaultdict(list)

        self.mintime = None
        self.maxtime = None
        self.timestep = None
        self.occ = None
        self.tf = default_trajectory_formatter

    def get_nonempty_interaction_list(self,include_distances=False):
        #
        # We never include the wat_resi, because we're just saying
        # what the interactions are.
        #
        # This is necessary for comparing to SSBTs
        # so, in the general case, we will not include distances.
        # However, it's also useful for building up other lists of
        # info, so we will allow the possibility of including
        # distances.
        #
        if include_distances:
            return [(self.resi1,self.resn1,self.atomname1,self.interaction1,dist1),
                    (self.resi2,self.resn2,self.atomname2,self.interaction2,dist2),
                    ]
        else:
            return [(self.resi1,self.resn1,self.atomname1,self.interaction1,),
                    (self.resi2,self.resn2,self.atomname2,self.interaction2,),
                    ]

    def __getitem__(self,time):
        return self._times[time]
    def __setitem__(self,time,wat_resi):
        if 0:
            print "setting",time,wat_resi
        self._times[time].append(wat_resi)
        return self._times[time]
    def __hash__(self):
        """
        SUPER IMPORTANT NOTE:
        It is very important to make sure that this hash function
        is the same as the hash function used for an SSBW.  Otherwise,
        the indexing into .trajectories won't work at all.

        We don't include distance for the same reason that we don't include
        wat_resi.
        """
        inter = [(self.resi1,self.resn1,self.atomname1,self.interaction1,),
                (self.resi2,self.resn2,self.atomname2,self.interaction2,),
                 ]
        inter.sort()
        if 0:
            print "Will hash..",tuple(inter),"to",hash(tuple(inter)),"in bwt"
        return hash(tuple(inter))
    def __repr__(self):
        try:
            verbose = False
            if verbose:
                result = '<BI %7.1fps occ: %5.1f adt: %5.1fps %3i [%s] (%3s %3s %-
4s):%13s, (%3s %3s %-4s):%13s,>'%(self.maxtime - self.mintime + self.timestep,

self.occ * 100,

self.average_dwell_time,

self.num_waters_seen,

self.__wat_resis,

self.resn1,self.resi1,self.atomname1,

self.interaction1,

self.resn2,self.resi2,self.atomname2,

self.interaction2,

)
            else:
                result = '<BI %7.1fps occ: %5.1f adt: %5.1fps %3i (%3s %3s %-4s):%13s,
(%3s %3s %-4s):%13s,>'%(self.maxtime - self.mintime + self.timestep,

self.occ * 100,

self.average_dwell_time,

self.num_waters_seen,

#     self.__wat_resis,

self.resn1,self.resi1,self.atomname1,

self.interaction1,

self.resn2,self.resi2,self.atomname2,

self.interaction2,

)

        except TypeError:
            print "Args"
            print (self.maxtime, self.mintime, self.timestep,
                   self.occ,100,
                   self.average_dwell_time,
                   self.num_waters_seen,
                   self.resn1,self.resi1,self.atomname1,
                   self.interaction1,
                   self.resn2,self.resi2,self.atomname2,
                   self.interaction2,)
            raise
        return result
    def contains_resi(self,resis):
        """
        Returns true of anything in resis is contained in any of our legs.

        Parameters
        ----------

        resis: a single resi, a list of resis, or None.  If None, we will
               return True for everything.
        """
        if resis is None:
            return True
        if type(resis) == type(''):
            resis = [int(i) for i in resis.split()]
        elif type(resis) == type(1):
            resis = [resis,]
        if len(self.get_nonempty_interaction_list()) != 2:
            raise NotImplementedError("Only bwts of length two are supported")
        for resi in resis:
            if resi == self.resi1:
                return True
            if resi == self.resi2:
                return True
        return False
    def get_trajectory_string(self,numchunks):
        """
        TODO: FIXME: verify that this is correct

        """
        if numchunks is None:
            result = ''
            for t in range(self.mintime,self.maxtime+self.timestep,self.timestep):
                if self[t]:
                    #result += '.'
                    result += '%s,'%self[t]
                else:
                    result += ' '
        else:
            result = self.tf.format(self.mintime,
                                    self.maxtime,
                                    self.timestep,
                                    numchunks,
                                    self,
                                    )

        return result
    def finalize(self,combination_method,min_required_dwell_time,looseness):
        """
        Calculate occupancy, dwell times.

        Parameters
        ----------

        combination_method: We see several trajectories where the occupancy by a given
                            water molecule will look like ..... . . .....
                            If you choose method 'strict' you'll see that as the same
                            water molecule occupying the bridge 4 different times.
                            The average dwell time there would be 4+1+1+4/4.

                            If you choose method 'combine' we will simply keep track
                            of how much time a given water molecule spends in the
                            bridge for the entire simulation.  That fixes up the
                            above case, but has some trouble.  For instance, of
                            water molecule 3254 occupies the bridge for 4ps, then
                            leaves for 1ns, then comes back for 8ps, we'll record
                            a dwell time of 12ps.

                            The difference between these methods is very significant.
                            In one particular case, for instance, I see it change
                            the dwell time from 465ps to 37.4ps.

                            If you choose method 'loose' we will try to be a little
                            smarter about combining the small trajectories.  We will
                            allow a skip of one between them.  That is,.. ....  ...
                            will be turned into .......  ... which is, I think,
                            a little better.

        looseness: If we're using method 'loose', this tells us how many
                   snapshots can be missing and still let us combine the
                   trajectory.

        min_required_dwell_time: measured in timesteps, not picoseconds.

        """
        from collections import defaultdict
        #
        # Occupancy
        #
        occ = 0.
        r = range(self.mintime,self.maxtime+self.timestep,self.timestep)
        for t in r:
            if self._times[t]:
                occ += 1
        self.occ = occ/len(r)

        #
        # Dwell times
        #
        #combination_method = 'strict'
        if combination_method == 'strict':
            raise NotImplementedError("'strict' method is no longer supported.  Please
look through the source and re-enable it if you want.")
            counts = []
            last_wat = None
            count = 1
            for t in r:
                if self._times[t] == last_wat:
                    count += 1
                else:
                    if last_wat not in (False,None,[]):
                        counts.append(count)
                    count = 1
                last_wat = self._times[t]
            if last_wat != None:
                counts.append(count)
            self.dwell_times = [c * self.timestep for c in counts]
        elif combination_method == 'combine':
            raise NotImplementedError("'combine' method is no longer supported.  Please
look through the source and re-enable it if you want.")
            counts = defaultdict(int)
            for t in r:
                wat_resi = self._times[t]
                if wat_resi not in (False,None,[]):
                    counts[wat_resi] += 1
            self.dwell_times = [i * self.timestep for i in counts.values()]
        elif combination_method == 'loose':
            """
            Writing this algorithm myself is actually a little tricky.  Instead,
            I'm just going to use strings.  For each water that we see, I'll
            make a trajectory string that contains 'x' when the water is there and
            ' ' when it's not.  Then, I can use string.replace('x x', 'xxx').
            """
            self.dwell_times = []
            #
            # I used to say this:
            #wat_resis = [i for i in set(self._times.values()) if i not in
(None,False,[])]
            # but that doesn't work anymore.  We've made self._times
            # a list that contains all of the wat_resis satisfying this
            # particular bridge at a given time.
            #
            wat_resis = set()
            for i in self._times.values():
                for j in i:
                    wat_resis.add(j)
            for wat_resi in wat_resis:
                traj_str = ''
                for t in r:
                    if wat_resi in self._times[t]:
                        traj_str += 'x'
                    else:
                        traj_str += ' '
                #print wat_resi,traj_str
                """
                A Note About Regular Expressions:

                This is not good enough:

                for i in range(1,looseness+1):
                    traj_str = traj_str.replace('x'+' '*i+'x','x'+'x'*i+'x')
                    print wat_resi,traj_str,'replacing',i

                because the regular expression engine returns only non-overlapping
                matches.  That means that we'll see something like this:

                In [71]: 'x x xx  x'.replace('x x','xxx')
                Out[71]: 'xxx xx  x'

                In order to take care of overlapping matches, we have to apply the
                pattern multiple times.  Question: how many times?  I think that
                two is enough, because the pattern and replacement is simple enough
                that we really only care about its edges.

                Who really cares about that, though?  I'll just keep replacing until
                I stop finding what I'm looking for.
                """
                for i in range(1,looseness+1):
                    pattern     = 'x'+' '*i+'x'
                    replacement = 'x'+'x'*i+'x'
                    while traj_str.find(pattern) >= 0:
                        traj_str = traj_str.replace(pattern,replacement)
                        #print wat_resi,traj_str,'replacing',i
                for traj in traj_str.split():
                    self.dwell_times.append(len(traj) * self.timestep)
        self.dwell_times = [i for i in self.dwell_times if i >=
self.timestep*min_required_dwell_time]
```

```
        self.num_waters_seen = len(self.dwell_times)
        self.average_dwell_time = 0
        if self.dwell_times:
            self.average_dwell_time = sum(self.dwell_times)/len(self.dwell_times)
        self.__wat_resis=wat_resis #TODO: FIXME: make sure this actually works
        #
        # TODO: clarify this: there's a question about exactly what I mean by
"num_waters_seen"
        # as is, it's the number of .. well, make sure it makes sense later, with 2
molecules at once, etc.
        #

        #
        # This is a little flakey .. if we're there for 10 steps, gone for one
        # and back for 10, that will show up as seen twice.
        #




class SingleSnapshotBridgingWater:

    def __init__(self,
                 wat_resi,
                 (resi1,resn1,atomname1,interaction1,dist1,_angle1,angle1),
                 (resi2,resn2,atomname2,interaction2,dist2,_angle2,angle2),

(resi3,resn3,atomname3,interaction3,dist3,_angle3,angle3)=(None,None,None,None,None,None,
None,),

(resi4,resn4,atomname4,interaction4,dist4,_angle4,angle4)=(None,None,None,None,None,None,
None,),

(resi5,resn5,atomname5,interaction5,dist5,_angle5,angle5)=(None,None,None,None,None,None,
None,),

(resi6,resn6,atomname6,interaction6,dist6,_angle6,angle6)=(None,None,None,None,None,None,
None,),

(resi7,resn7,atomname7,interaction7,dist7,_angle7,angle7)=(None,None,None,None,None,None,
None,),

(resi8,resn8,atomname8,interaction8,dist8,_angle8,angle8)=(None,None,None,None,None,None,
None,),
                 ):
        """

        Parameters
        ----------
        resi1,atomname1,interaction1,dist1: one of the residues, and in interaction
                                            in {donatingto,acceptingfrom}.
                                            153,donatingto would mean that the water is
                                            donating an hbond to residue 153 with
distance dist1.

        At the moment, there's no reason to record the angles, so we won't.

        """
        self.wat_resi = wat_resi
        if resi1 is not None: resi1 = int(resi1)
        if resi2 is not None: resi2 = int(resi2)
        if resi3 is not None: resi3 = int(resi3)
        if resi4 is not None: resi4 = int(resi4)
        if resi5 is not None: resi5 = int(resi5)
        if resi6 is not None: resi6 = int(resi6)
        if resi7 is not None: resi7 = int(resi7)
        if resi8 is not None: resi8 = int(resi8)
        self.interactions = {(resi1,resn1,atomname1,dist1):interaction1,
                             (resi2,resn2,atomname2,dist2):interaction2,
                             (resi3,resn3,atomname3,dist3):interaction3,
                             (resi4,resn4,atomname4,dist4):interaction4,
                             (resi5,resn5,atomname5,dist5):interaction5,
                             (resi6,resn6,atomname6,dist6):interaction6,
                             (resi7,resn7,atomname7,dist7):interaction7,
                             (resi8,resn8,atomname8,dist8):interaction8,
                            }
    def get_nonempty_interaction_list(self,include_distances=False):
        #
        # We never include the wat_resi, because we're just saying
        # what the interactions are.
        #
        # This is necessary for comparing to BWTs
        # so, in the general case, we will not include distances.
        # However, it's also useful for building up other lists of
        # info, so we will allow the possibility of including
        # distances.
        #
        result = []
        for k,v in self.interactions.iteritems():
            if None in k:
                continue
            if v is None:
                continue
            if include_distances:
                result.append((k[0],k[1],k[2],v,k[3]))
            else:
                result.append((k[0],k[1],k[2],v))
        return result

    def __repr__(self):
        """

        string representation.  For purposes of principal, we will not include
        the bridging water here.
        """
        #result = '(%s,{'%self.wat_resi
        result = '<SSBW %s '%self.wat_resi
        for k,v in self.interactions.iteritems():
            if None in k:
                continue
            if v is None:
                continue
            result += "%s:'%s',"%(k,v)
        #result +='})'
        result +='>'
        return result
        #return str((self.wat_resi,self.interactions))
    def __hash__(self):
        """
        """
        #
        # if we get rid of self.wat_resi, we may not be able to
        # correctly keep track of dwell times for particular waters.
        #
        inter = self.get_nonempty_interaction_list()
        inter.sort()
        #return hash((self.wat_resi,tuple(inter)
        #             ))
        if 0:
            print "Will hash",inter
        return hash(tuple(inter))

    def __eq__(self,other):
        if 0:
            print "Comparing",self,other,sorted(self.get_nonempty_interaction_list()) ==
sorted(other.get_nonempty_interaction_list())
        return sorted(self.get_nonempty_interaction_list()) ==
sorted(other.get_nonempty_interaction_list())
        #return self.interactions == other.interactions
    def __ne__(self,other):
        return sorted(self.get_nonempty_interaction_list) !=
sorted(other.get_nonempt_interaction_list)
        #return self.interactions != other.interactions


class SingleSnapshotHbondEmitter:
    def __init__(self,states,hbond_dist_cutoff=3.5,hbond_angle_cutoff=None):
        '''

        Parmeters
        ---------

        states: the number of states we will analyze.

        hbond_dist_cutoff: HeavyAtom-Hydro distance, so we default to a generous 3.5.

        hbond_angle_cutoff: At the moment, we will mimic ptraj and not have an angle
cutoff.

        Our lists of donors and acceptors are taken from
http://amber.scripps.edu/tutorials/basic/tutorial3/files/analyse_hbond.ptraj
        '''
        self.states = states
        self.hbond_dist_cutoff = hbond_dist_cutoff
        #if hbond_angle_cutoff is not None:
        #    raise NotImplementedError('hbond_angle_cutoff not implemented yet (we are
like ptraj default here).')
        self.hbond_angle_cutoff = hbond_angle_cutoff

        self.logfilename = 'HbondEmitterLogFile.txt'
    def log(self,message):
        f = file(self.logfilename,'a')
        f.write(message)
        f.close()
        sys.stdout.write(message)
        sys.stdout.flush()

    def create_selections(self):

        from pymol import cmd,stored
        import hbond_definitions
        hbond_definitions.do_standard_selections()

        if 0:
            #Once we do the per-state selections, we may be able
            # to resurrect this.
            cmd.select('prot_acceptors','prot_acceptors and %s'%sel)
            cmd.select('prot_donors','prot_donors and %s'%sel)
    def analyze_and_emit(self,outf):
        #
        # Note, I think it's a little clearer this way.  OTOH, we could
        # quite easily change the format so that analyze() takes in a
        # file handle and writes each bridge out to it on one line,
        # something like (state,bridge)
        #
        results = self.analyze()
        outf.write(str(results))

    def analyze(self):
        """
        go through the whole thing at once

        You might be wondering why I have all of these safety checks in here.
        It turns out that they've all been necessary at one time or another.

        In particular, I often find that the last water in a trajectory will
        trigger the PROBLEM WITH errors, and I'll often see things like
        FAILED H2DIST 3 1, neither of which should ever happen.
        """
        from pymol import cmd,stored
        self.create_selections()
        bridges = {}
        for state in range(1,self.states+1):
            bridges[state] = {}
            stored.wat_list = []
            #
            # Here, we have a question about how to make this as fast as
            # possible. After talking to Warren, it looks like the best thing
            # to do is to make a new object (tmp_wat_papd .. temporary waters +
            # protein acceptors + protein donors) that contains only the atoms
            # from a particular state that we are interested in. This sort of
            # object creation should be faster than passing the "state" argument
            # to get_distance and get_angle.
            #
            # We'll be a little bit generous in creating that object: we'll grab
            # everything within dist_cutoff of any relevant atoms, rather than
            # just searching for the heavy-heavy distances.
            #
            cmd.create('tmp_wat_papd','(prot_donors or prot_acceptors or neighbor
prot_donors) or (byres (resn WAT+HOH) within %s of (prot_donors or prot_acceptors or
neighbor prot_donors))'%(self.hbond_dist_cutoff),state,1)
            cmd.select('tmp_wat_near','byres (tmp_wat_papd and (resn WAT+HOH)) within %s
of (tmp_wat_papd in (prot_donors or prot_acceptors or neighbor
prot_donors))'%(self.hbond_dist_cutoff+0.1))
            cmd.iterate('tmp_wat_near and elem o','stored.wat_list.append(resi)')
            num_wat_resis = len(stored.wat_list)
            self.log('\nI will loop over %s nearby waters for state
%s\n'%(num_wat_resis,state))
            for (wat_idx,w_resi) in enumerate(stored.wat_list):
                pct_done = int(100*(wat_idx+1)/num_wat_resis)
                if divmod(pct_done,10)[-1] == 0: self.log('X')
                else: self.log('.')

                bridges[state][w_resi] = []
                stored.pa_list = []
                cmd.iterate('(tmp_wat_papd in prot_acceptors) within %s of (tmp_wat_papd
and resi %s and elem
h)'%(self.hbond_dist_cutoff,w_resi),'stored.pa_list.append((resi,resn,name))')
                for (resi,resn,name) in stored.pa_list:
                    # Here, we're looking for distances to the water hydrogens,
                    # which is why we use resi <w_resi> and name H1|H2.
                    #
                    # Also, the select(left, (not left) and (neighbor left))
                    # business is because our hbond definition file lists
                    # the hydrogens, but we want heavy-atom distances.

                    # Calculate the distance/angle w.r.t the water's H1
                    left_hydro = cmd.select('left_hydro',"resi %s and name H1 and
tmp_wat_papd"%w_resi)
                    if left_hydro != 1: self.log("PROBLEM WITH resi %s and name H1 and
tmp_wat_papd in state %s"%(w_resi,state))

                    left_heavy = cmd.select('left_heavy','(not left_hydro) and (neighbor
left_hydro) and tmp_wat_papd')
                    if left_heavy != 1: self.log("PROBLEM WITH new left sele resi %s and
name H1 and tmp_wat_papd in state %s"%(w_resi,state))

                    right = cmd.select('right',"resi %s and name %s and
tmp_wat_papd"%(resi,name))
                    if right != 1: self.log("PROBLEM WITH resi %s and name %s and
tmp_wat_papd in state %s"%(resi,name,state))

                    try:
                        h1dist = cmd.get_distance('left_heavy','right') # state not
included because we're in tmp_wat_papd
                    except:
                        self.log("FAILED H1DIST %s %s"%(left_heavy,right))
                        cmd.delete('adist')
                        h1dist = cmd.distance('adist','left_heavy','right')
                        self.log("DIST SAYS %s"%h1dist)
                    try:
                        h1angle = cmd.get_angle('left_heavy','left_hydro','right')
                    except:
                        self.log('FAILED H1ANGLE %s %s %s'%(left_heavy,left_hydro,right))
                        cmd.delete('anangle')
                        h1angle = cmd.angle('anangle','left_heavy','left_hydro','right')

                    # Calculate the distance/angle w.r.t the water's H2
                    left_hydro = cmd.select('left_hydro',"resi %s and name H2 and
tmp_wat_papd"%w_resi)
```

```
                    if left_hydro != 1: self.log("PROBLEM WITH resi %s and name H2 and
tmp_wat_papd in state %s(%s)"%(w_resi,state,left))

                    left_heavy = cmd.select('left_heavy','(not left_hydro) and (neighbor
left_hydro) and tmp_wat_papd')
                    if left_heavy != 1: self.log("PROBLEM WITH new left sele resi %s and
name H2 and tmp_wat_papd in state %s"%(w_resi,state))

                    right = cmd.select('right',"resi %s and name %s and
tmp_wat_papd"%(resi,name))
                    if right != 1: self.log("PROBLEM WITH resi %s and name %s and
tmp_wat_papd in state %s(%s)"%(resi,name,state,right))

                    try:
                        h2dist = cmd.get_distance('left_heavy','right') # state not
included because we're in tmp_wat_papd
                    except:
                        self.log("FAILED H2DIST %s %s"%(left,right))
                        cmd.delete('adist')
                        h2dist = cmd.distance('adist','left_heavy','right')
                        self.log("DIST SAYS %s"%h2dist)
                    try:
                        h2angle = cmd.get_angle('left_heavy','left_hydro','right')
                    except:
                        self.log('FAILED H2ANGLE %s %s %s'%(left_heavy,left_hydro,right))
                        cmd.delete('anangle')
                        h2angle = cmd.angle('anangle','left_heavy','left_hydro','right')

                    # Given the choice, we chose the one with the smallest distance.
                    if h1dist <= h2dist:
                        if (h1dist <= self.hbond_dist_cutoff) and (h1angle >=
self.hbond_angle_cutoff):

bridges[state][w_resi].append((resi,resn,name,'donatingto',h1dist,'angle',h1angle))
                        elif (h2dist <= self.hbond_dist_cutoff) and (h2angle >=
self.hbond_dist_cutoff):

bridges[state][w_resi].append((resi,resn,name,'donatingto',h2dist,'angle',h2angle))
                    else:
                        if (h2dist <= self.hbond_dist_cutoff) and (h2angle >=
self.hbond_dist_cutoff):

bridges[state][w_resi].append((resi,resn,name,'donatingto',h2dist,'angle',h2angle))
                        elif (h1dist <= self.hbond_dist_cutoff) and (h1angle >=
self.hbond_angle_cutoff):

bridges[state][w_resi].append((resi,resn,name,'donatingto',h1dist,'angle',h1angle))
                    # old dist-only code:
                    #if (h2dist <= self.hbond_dist_cutoff) and (h2angle >=
self.hbond_dist_cutoff):
                    #
bridges[state][w_resi].append((resi,resn,name,'donatingto',min(h1dist,h2dist)))
                stored.pd_list = []
                cmd.iterate('(tmp_wat_papd in prot_donors) within %s of (tmp_wat_papd and
resi %s and elem
o)'%(self.hbond_dist_cutoff,w_resi),'stored.pd_list.append((resi,resn,name))')
                for (resi,resn,name) in stored.pd_list:
                    left = cmd.select('left',"resi %s and name o and
tmp_wat_papd"%w_resi)
                    if left != 1: self.log("PROBLEM WITH resi %s and name o and
tmp_wat_papd in state %s(%s)"%(w_resi,state,left))

                    right_hydro = cmd.select('right_hydro',"resi %s and name %s and
tmp_wat_papd"%(resi,name))
                    if right_hydro != 1: self.log("PROBLEM WITH resi %s and name %s and
tmp_wat_papd in state %s(%s)"%(resi,name,state,right))

                    right_heavy = cmd.select('right_heavy','(not right_hydro) and
(neighbor right_hydro) and tmp_wat_papd')
                    if right_heavy != 1: self.log("PROBLEM WITH new right sele resi %s
and name %s and tmp_wat_papd in state %s(%s)"%(resi,name,state,right))

                    try:
                        hdist = cmd.get_distance('left','right_heavy') # state not
included because we're in tmp_wat_papd
                    except:
                        self.log("FAILED HDIST %s %s"%(left,right))
                        cmd.delete('adist')
                        hdist = cmd.distance('adist','left','right_heavy')
                        self.log("DIST SAYS %s"%hdist)
                    try:
                        hangle = cmd.get_angle('left','right_hydro','right_heavy')
                    except:
                        self.log('FAILED HANGLE %s %s %s'%(left,right_hydro,right_heavy))
                        cmd.delete('anangle')
                        hangle = cmd.angle('anangle','left','right_hydro','right_heavy')
                    if (hdist <= self.hbond_dist_cutoff) and (hangle >=
self.hbond_angle_cutoff):

bridges[state][w_resi].append((resi,resn,name,'acceptingfrom',hdist,'angle',hangle))
        real_bridges = {}
        for state in bridges:
            for wat_resi,bridge in bridges[state].iteritems():
                if len(bridge) >= 2:
                    if state not in real_bridges: real_bridges[state] = []
                    real_bridges[state].append([wat_resi,] + bridge)
        return real_bridges


def
find_bridging_waters_in_trajectory(name,start,stop,hbond_dist_cutoff,hbond_angle_cutoff,o
verwrite=False):
    """
    Read in a trajectory file and find bridging hbonds.

    Parameters
    ----------

    name: We will read in <name>.trj and <name>.top from the
          current working directory.

    start,stop: The states of the trajectory are like indices in
                a list.  We will process list[start:stop].  You
                should know that there is no state 0.

    overwrite: If this is False, we will not overwrite existing files.
               Rather, we will print a message and do nothing.

    Output
    ------

    The results will be written to <name>_hbonds_<start>_<stop>.txt.
    If that file exists, it will be overwritten.  That file will
    contain a Python dictionary of all of the bridging waters.  The
    keys in that dictionary will be state numbers, relative to the
    start:stop range.  That is, if it's the first state we actually
    read in after all of the skipping, it'll be called state 1.
    That's why the emitter doesn't have to know what the absolute
    starts and stops are.
    """
    fname = '%s_hbond_%s_%s.txt'%(name,start,stop)
    if os.path.exists(fname) and not overwrite:
        print fname,"already exists.  Skipping."
        return
    print "Opening",fname,"for writing"
    f = file(fname,'w')

    from pymol import cmd,stored
    cmd.delete('all')
    print "ready to load top"
    cmd.load(name+'.top')
    print "ready to load trj"
    cmd.load_traj(name+'.trj',start=start,stop=stop)
    #
    # Yeah, tell me about it. But, PyMOL will often load up water molecules from
    # an AMBER trajectory so that the hydrogens are bonded to each other.
    #
    cmd.unbond('hydro','hydro')
    states = cmd.count_states(name)
    e =
SingleSnapshotHbondEmitter(states=states,hbond_dist_cutoff=hbond_dist_cutoff,hbond_angle_
cutoff=hbond_angle_cutoff)
    e.analyze_and_emit(f)
    print "Closing",fname
    f.close()

if __name__ == '__pymol__':
    pymol.cmd.extend('fbwt',find_bridging_waters_in_trajectory)

def
get_hbond_trajectories(structure,timestep,fnames,combination_method,min_required_dwell_ti
me,looseness,dist_cutoff,angle_cutoff):
    """
    Takes in structure,timestep,fnames, returns a BridgingWaterTrajectoryAnalyzer.

    This assumes that times can be automatically determined from
    filenames.
    """
    print "Bridging Water Trajectory calculated with combination_method %s, looseness %s,
min_required_dwell_time %s, dist_cutoff %s, angle_cutoff %s"%(combination_method,

looseness,

min_required_dwell_time,

dist_cutoff,

angle_cutoff,

)
    a = BridgingWaterTrajectoryAnalyzer(structure,timestep)
    #
    # There are two places where we filter things out.
    # 1) When we're reading in the files, we filter by static information
    #    like dist_cutoff and angle_cutoff
    # 2) When we're finalizing, we filter by trajectory information
    #    like min_required_dwell_time.
    #
    for fname in fnames:
        a.read_in_file(fname,dist_cutoff=dist_cutoff,angle_cutoff=angle_cutoff)
    a.finalize(combination_method=combination_method,
               min_required_dwell_time=min_required_dwell_time,
               looseness=looseness)
    return a

if __name__ == '__main__':
    #
    # Main code moved to script in example directory.
    # Put some test functions here.
    #
    pass
```

Directory listing 2

A separate PyPAT installation provides the advanced features for combining hydrogen-bond output files.

```
phar20-187~/work/PyPAT_hbonding/$ ls
combine_hbonds.py
residue_lists
tool_utils.py
compare_hbonds.py
```

```
md_analysis_utils.py
setup_hbond_ptraj.py
hbond_analysis_utils.py
subset_hbonds.py
```

Files 2

combine_hbonds.py

```
#!/usr/bin/env python

'''
Combine hbond output from a series of ptraj runs into one data set.
'''

import sys, os
from optparse import OptionParser
from md_analysis_utils import get_resinum_to_resi_map
from tool_utils import parse_residue_list, parse_atom_list
from hbond_analysis_utils import combine_hbonds

if __name__ == '__main__':

    usage = """%prog FILE1 [ FILE2 [ ... ] ] [options]
FILE1 and additional optional FILEs are files containing hbond data that
were produced by the ptraj hbond command.  At least one such file is
required.  The data in the files are spliced together to created a
unified data set.

Important notes:
* An AMBER prmtop or a PDB file may be input with the -p option.  The
  file will be used to determine the residue name associated with each
  residue number in the system.  If the file name does not end with
  '.pdb', the file will be assumed to be an AMBER prmtop file.  The
  offset and amino acid code are also used in the residue name
  generation.
* The resi_criteria option takes a comma-separated string containing any
  or all of the following:
        - individual residue numbers
        - a range of numbers, separated by a '-'
        - strings associated with valid residue lists in the
            standard file 'residue_lists'
* The atom_criteria option takes a comma-separated string containing
  the atom names to report.  In addition, the string can contain any
  of these strings:
       - 'bb_only': only H-bonds between two backbone atoms
       - 'not_bb': no H-bonds between two backbone atoms
       - 'protein_only': no H-bonds involving water"""

    # Parse command line options

    parser = OptionParser(usage = usage)

    parser.add_option('-o', '--output-file',
           dest = 'output_file',
           help = 'The name of the output file.  If None, the results will be written to
stdout.  Supplying a name is recommended.  [default: %default]')
    parser.add_option('-B', '--hbond-data-dir',
           dest = 'hbond_data_dir',
           help = 'The directory that contains the H-bond data files.  If None, the file
names will be used without modification, and output will be written to the current
directory.  [default: %default]')
    parser.add_option('-g', '--segment-size', type = 'int',
           dest = 'segment_size', default = 1000,
           help = 'The number of frames included in each segment of the trajectory.
[default: %default]')
    parser.add_option('-p', '--prmtop-file',
           dest = 'prmtop_file',
           help = 'Amber parameter/topology file or PDB file for the system.  [default:
%default]')
```

```
    parser.add_option('-D', '--prmtop-dir',
           dest = 'prmtop_dir', default = None,
           help = 'The directory that contains the prmtop (or PDB) file.  If None, the
name of the file will be used without modification.  [default: %default]')
    parser.add_option('-r', '--resi-offset', type = 'int',
           dest = 'resi_offset', default = 0,
           help = 'The offset between the residue numbers in the prmtop (or PDB) file and
the actual residue numbers.  [default: %default]')
    parser.add_option('-a', '--aa-code', type = 'int',
          dest = 'aa_code', default = 3,
          help = 'Must be 1 or 3.  Indicates the use of 1- or 3-letter amino acid codes
in residue names.  [default: %default]')
    parser.add_option('-y', '--occ-graph-only', action = 'store_true',
           dest = 'occ_graph_only', default = False,
           help = 'Flag to report only the occupancy and graph data (no distance or angle
data).  [default: %default]')
    parser.add_option('-R', '--resi-criteria',
           dest = 'resi_criteria', default = 'all',
           help = 'A comma- and dash-separated list of residue numbers to include in the
analysis.  [default: %default]')
    parser.add_option('-A', '--atom-criteria',
           dest = 'atom_criteria', default = 'all',
           help = 'A comma-separated list of atom names to include in the analysis.
[default: %default]')
    parser.add_option('-O', '--occ-thresh', type = 'float',
           dest = 'occ_thresh', default = 0.0,
           help = 'The minimum occupancy threshold that the H-bonds must have to be
reported.  [default: %default]')

    options, hbond_files = parser.parse_args()

    # Process options

    if options.aa_code != 1 and options.aa_code != 3:
       print 'Warning:  Illegal amino acid code.  Must be 1 or 3.\n' + \
             '  Will use the default of 3.'
       options.aa_code = 3

    if options.prmtop_dir != None:
       options.prmtop_file = os.path.join(options.prmtop_dir, options.prmtop_file)
    resi_map = get_resinum_to_resi_map(options.prmtop_file,
               offset = options.resi_offset, aa_code = options.aa_code)

    resi_criteria = parse_residue_list(options.resi_criteria)
    if not resi_criteria:
       sys.exit('ERROR:  No residues selected in residue string.\n')
    atom_criteria = parse_atom_list(options.atom_criteria)

    # Perform function

    combine_hbonds(hbond_files = hbond_files,
       segment_size = options.segment_size,
       resi_map = resi_map,
       output_file = options.output_file,
       resi_criteria = resi_criteria,
       atom_criteria = atom_criteria,
       occ_thresh = options.occ_thresh,
       occ_graph_only = options.occ_graph_only,
       hbond_data_dir = options.hbond_data_dir  )
```

residue_lists

```
# Lists for DHFR

# Lists for BACE1
flap: 67-75
10s: 9-14
BACE1-loops: flap,10s
```

tool_utils.py

```
#!/usr/bin/env python

import sys, os

def parse_residue_list_file(filename = None):
    '''
    Receives a file name and parses that file to create a dictionary
    of residue lists associated with the lists' names, which can then
    be used in function parse_residue_list().

    Although it looks like 'None' is the default for the filename
    keyword, the function will look for a file named 'residue_lists'
    as the default.  The reason for the None is that the warning
    message output by this function if the file was not found is
    suppressed if no name was actually passed to the function.

    The 'residue_lists' file should contain lines with the following
    pattern:
    NAME : RESIDUE_STRING
    where NAME is the name to be associated with the residue list
    and RESIDUE_STRING is a comma-separated string that can be
    parsed to create a list of residue names.  The residue string
    may contain any or all of the following:
        * individual residue numbers
        * sequences of numbers, with the first and last numbers
            separated by a '-'
        * strings associated with residue lists that have
            already been defined in the file
    '''
    if not filename:
        filename = 'residue_lists'
        warning_str = ''
    else:
        warning_str = 'Warning:  Could not open file ' + filename + \
                   '.\n  No standard residue lists are available.'

    try:
        f = file(filename)
    except:
        print warning_str
        return {}

    standard_residue_lists = {}

    for line in f:
        if not line.strip() or line.strip().startswith('#'):
            continue
        try:
            resi_list = []
            key, string_to_parse = line.split(':')
            key = key.strip()
            for piece in string_to_parse.strip().split(','):
                if standard_residue_lists.has_key(piece):
                    resi_list.extend(standard_residue_lists[piece])
                elif '-' in piece:
                    first, last = piece.split('-')
                    resi_list.extend(range(int(first), int(last) + 1))
                else:
                    resi_list.append(int(piece))
        except:
            print 'Warning:  Ignoring line in file ' + filename + '\n' + \
                  '  because of improper syntax:\n' + line
        resi_list = sorted(list(set(resi_list)))
        standard_residue_lists[key] = resi_list

    return standard_residue_lists
```

```
def parse_residue_string(string_to_parse, offset = 0, filename = None):
    '''
    Receives a comma-separated string and returns a list of residues.
    The string may contain any or all of the following:
        * individual residue numbers
        * a range of numbers, separated by a '-'
        * strings associated with valid residue lists in the
            standard file 'residue_lists' (a different file name
            can be passed to this function using the filename keyword)
    A list of individual residue numbers will be printed out.

    Option offset can be used to create, e.g., a zero-based list
    for indexing.
    '''
    standard_residue_lists = parse_residue_list_file(filename)

    piece_list = string_to_parse.strip().split(',')
    if 'all' in piece_list:
        return ['all']

    try:
        resi_list = []
        for piece in piece_list:
            if standard_residue_lists.has_key(piece):
                for resi in standard_residue_lists[piece]:
                   resi_list.append(resi - offset)
            elif '-' in piece:
                first, last = piece.split('-')
                resi_list.extend(range(int(first) - offset, int(last) + 1 - offset))
            else:
                resi_list.append(int(piece) - offset)
    except:
        print 'Warning:  Residue list string ' + string_to_parse + \
              ' has improper syntax.'
        return []

    resi_list = sorted(list(set(resi_list)))

    output_str = 'Residue list ' + string_to_parse + ' represents ' + \
                 str(len(resi_list)) + ' residues:\n'
    for resi in resi_list:
        output_str += str(resi) + ','
    output_str = output_str[:-1] + '\n'
    print output_str

    return resi_list


def parse_residue_list(string_to_parse, offset = 0, filename = None):
    '''
    Receives a string and returns one or two lists of residues.  If
    the string contains a ':', the strings on both sides of the colon
    are parsed by parse_residue_string and the two lists are returned.
    If no colon is present, the string itself is parsed and a single
    list is returned.
    '''
    string_pieces = string_to_parse.strip().split(':')

    if len(string_pieces) == 1:
        return parse_residue_string(string_pieces[0], offset, filename)
    elif len(string_pieces) == 2:
        return parse_residue_string(string_pieces[0], offset, filename), \
               parse_residue_string(string_pieces[1], offset, filename)
    else:
        print 'Warning:  Residue list string ' + string_to_parse + \
              ' has improper syntax.'
        return []


def parse_atom_list(string_to_parse):
    '''
    Receives a string and splits by commas.  Used to parse the list of
    atoms given to various hbond programs.
    '''
    piece_list = string_to_parse.strip().split(',')
    return piece_list
```

compare_hbonds.py

```
#!/usr/bin/env python

'''
Set up ptraj input files to run a series of H-bond calculations.
'''

import sys, os
from optparse import OptionParser
from tool_utils import parse_residue_list, parse_atom_list
from hbond_analysis_utils import compare_hbonds

if __name__ == '__main__':

    usage = """%prog FILE1 [ FILE2 [ ... ] ] [options]
FILE1 and additional optional files are files created by
combine_hbonds.py that contain datasets of H-bonds from different
trajectories of the same system.  The particular subset of the H-bonds
and the metric for sorting can be specified by the user.

Important notes:
* Use the -i option to provide meaningful identifiers for the different
  trajectories.
* The resi_criteria option takes a comma-separated string containing any
  or all of the following:
        - individual residue numbers
        - a range of numbers, separated by a '-'
        - strings associated with valid residue lists in the
          standard file 'residue_lists'
* The atom_criteria option takes a comma-separated string containing
  the atom names to report.  In addition, the string can contain any
  of these strings:
        - 'bb_only': only H-bonds between two backbone atoms
        - 'not_bb': no H-bonds between two backbone atoms
        - 'protein_only': no H-bonds involving water
* If the H-bonds are sorted by occ_pct (the occupancy percentage), any
  H-bond that has an occupancy greater than the occ_thresh value will
  be retained.  If the H-bonds are sorted by occ_diff (the difference
  between the largest and smallest occupancy percentages for the
  systems), donor, or acceptor, only those H-bonds with occ_diff
  greater than the occ_thresh value will be retained.
* Note that ptraj uses a definition of H-bond donor and acceptor that is
  opposite of the normal convention.  This program follows the
  definitions of ptraj, in which the acceptor is the atom covalently
  bonded to the hydrogen atom."""

    # Parse command line options

    parser = OptionParser(usage = usage)

    parser.add_option('-o', '--output-file',
           dest = 'output_file',
           help = 'The name of the output file.  If None, the results will be written to
stdout.  [default: %default]')
    parser.add_option('-B', '--hbond-data-dir',
           dest = 'hbond_data_dir',
          help = 'The directory that contains the H-bond data files.  If None, the file
names will be used without modification, and output will be written to the current
directory.  [default: %default]')
    parser.add_option('-i', '--identifiers',
           dest = 'identifiers',
           help = 'Comma-separated list of identifying strings for the trajectories to be
compared.  If None, the trajectories will simply be numbered.  [default: %default]')
    parser.add_option('-s', '--sort',
           dest = 'sort', default = 'occ_diff',
           choices = ['occ_diff', 'occ_pct', 'donor', 'acceptor'],
           help = 'The quantity used to sort the results.  Must be one of "occ_diff"
(occupancy difference), "occ_pct" (occupancy percentage), "donor", or "acceptor".  The
occupancy difference is the difference between the highest and lowest occupancy
percentages for a particular H-bond in the different trajectories.  [default: %default]')
    parser.add_option('-c', '--compress', action = 'store_true',
           dest = 'compress', default = False,
           help = 'Flag to compress the H-bond graph.  [default: %default]')
    parser.add_option('-y', '--occ-graph-only', action = 'store_true',
           dest = 'occ_graph_only', default = False,
           help = 'Flag to report only the occupancy and graph data (no distance or angle
data).  [default: %default]')
    parser.add_option('-R', '--resi-criteria',
           dest = 'resi_criteria', default = 'all',
           help = 'A comma- and dash-separated list of residue numbers to include in the
analysis.  [default: %default]')
    parser.add_option('-A', '--atom-criteria',
           dest = 'atom_criteria', default = 'all',
           help = 'A comma-separated list of atom names to include in the analysis.
[default: %default]')
    parser.add_option('-O', '--occ-thresh', type = 'float',
           dest = 'occ_thresh', default = 0.0,
           help = 'The minimum occupancy threshold that the H-bonds must have to be
reported.  [default: %default]')

    options, hbond_files = parser.parse_args()

    # Process options

    identifiers = options.identifiers.split(',')

    resi_criteria = parse_residue_list(options.resi_criteria)
    if not resi_criteria:
        sys.exit('ERROR:  No residues selected in residue string.\n')
    atom_criteria = parse_atom_list(options.atom_criteria)

    # Perform function

    compare_hbonds(hbond_files = hbond_files,
       identifiers = identifiers,
       output_file = options.output_file,
       resi_criteria = resi_criteria,
       atom_criteria = atom_criteria,
       occ_thresh = options.occ_thresh,
       occ_graph_only = options.occ_graph_only,
       sort = options.sort,
       compress = options.compress,
       hbond_data_dir = options.hbond_data_dir   )
```

md_analysis_utils.py

```
#!/usr/bin/env python

"""
Contains tools for analyzing MD trajectories
"""

from string import upper, capitalize

def ThrLett_to_OneLett(resi, suppress_alert = True):
    """
    Usage:  ThrLett_to_OneLett(resi, suppress_alert = True)

    This function receives resi, the three-letter code
    for an amino acid and returns its one-letter code.
    If the three-letter code is not recognized, it is
    simply returned.  If suppress_alert is False,
    a message will be printed alerting the user that
    the residue was not recognized
    """

    resiu = upper(resi)
    if resiu == 'ALA':
        return 'A'
    elif resiu == 'ARG' or resiu == 'ARN':
        return 'R'
    elif resiu == 'ASN':
        return 'N'
    elif resiu == 'ASP' or resiu == 'ASH':
        return 'D'
    elif resiu == 'CYS' or resiu == 'CYX' or \
         resiu == 'CYM':
        return 'C'
    elif resiu == 'GLN':
        return 'Q'
    elif resiu == 'GLU' or resiu == 'GLH':
        return 'E'
    elif resiu == 'GLY':
        return 'G'
    elif resiu == 'HIS' or resiu == 'HIE' or \
         resiu == 'HID' or resiu == 'HIP':
        return 'H'
    elif resiu == 'ILE':
        return 'I'
    elif resiu == 'LEU':
        return 'L'
    elif resiu == 'LYS' or resiu == 'LYN':
        return 'K'
    elif resiu == 'MET':
        return 'M'
    elif resiu == 'PHE':
        return 'F'
    elif resiu == 'PRO':
        return 'P'
    elif resiu == 'SER':
        return 'S'
    elif resiu == 'THR':
        return 'T'
    elif resiu == 'TRP':
        return 'W'
    elif resiu == 'TYR':
        return 'Y'
    elif resiu == 'VAL':
        return 'V'
    else:
        if not suppress_alert:
            print resi, "not recognized as residue.  Returning", resi
        return resi


def OneLett_to_ThrLett(resi, cap = 'standard', suppress_alert = True):
    """
    Usage:  OneLett_to_ThrLett(resi, cap = 'standard',
               suppress_alert = True)

    This function receives resi, a one-letter amino acid
    code, and returns the three letter amino acid code.
    If cap(italization) is standard, then only the first
    letter will be capitalized.  If cap is all, then all
    letters will be capitalized (as in a PDB file).  If
    the one-letter amino acid is not recognized, it is
    simply returned.  If option suppress_error is False,
    a message will also be printed to alert to user that
    the code was not recognized.
    """
    if resi == 'A':
        res3 = 'Ala'
    elif resi == 'C':
        res3 = 'Cyx'
    elif resi == 'D':
        res3 = 'Asp'
    elif resi == 'E':
        res3 = 'Glu'
    elif resi == 'F':
        res3 = 'Phe'
    elif resi == 'G':
        res3 = 'Gly'
    elif resi == 'H':
        res3 = 'His'
    elif resi == 'I':
        res3 = 'Ile'
    elif resi == 'K':
        res3 = 'Lys'
    elif resi == 'L':
        res3 = 'Leu'
    elif resi == 'M':
        res3 = 'Met'
    elif resi == 'N':
        res3 = 'Asn'
    elif resi == 'P':
        res3 = 'Pro'
    elif resi == 'Q':
        res3 = 'Gln'
    elif resi == 'R':
        res3 = 'Arg'
    elif resi == 'S':
        res3 = 'Ser'
    elif resi == 'T':
        res3 = 'Thr'
    elif resi == 'V':
        res3 = 'Val'
    elif resi == 'W':
        res3 = 'Trp'
    elif resi == 'Y':
        res3 = 'Tyr'
    else:
        if not suppress_alert:
            print resi, "not recognized as residue.  Returning", resi
        res3 = resi

    if cap == 'all':
        res3 = upper(res3)
    elif cap != 'standard':
        print 'Ignoring invalid option for cap:', cap

    return res3


def get_resinum_to_resi_map(resiname_file, offset = 0, indexing = 1, aa_code = 3):
    """
    This function returns a dictionary that relates the residue
    number for a given system to a residue name.  A PDB or prmtop file
    for the system of interest is required.  The string that comprises
    the residue name (the values of the dictionary) is the amino
    acid code (in three-letter or one-letter format; default three-
    letter) followed by the residue number (e.g., Thr72 or T72).

    An optional offset can be passed to the function.  This
    changes the residue number in the string, which is useful if
    the numbering in the PDB file is not the preferred numbering
    for the system.

    For most indexed applications, the first residue is
    numbered as 0 instead of 1.  A zero-based dictionary can be
    set by passing indexing = 0 to the function.  The default
    is 1.

    If the PDB file input to the function is not
    found, a message will alert the user.  In this case, the
    values in the dictionary are simply the numbers from 1 to
    9999 (the maximum residue number of a PDB file).
    """
    resi_map = {}
    if resiname_file == None:
        print 'Warning:  No prmtop or PDB file given.\n' + \
              '  No residue number information will be presented.'
        for i in range(10000):
            resi_map[i] = str(i)
        return resi_map

    try:
        f = file(resiname_file)
    except IOError:
        print 'Warning:  Could not open ' + resiname_file + '.\n' + \
              '  No residue number information will be presented.'
        for i in range(10000):
            resi_map[i] = str(i)
        return resi_map

    # If the file is a prmtop file...

    if not resiname_file.endswith('.pdb'):
        resi_num = 1
        residue_section = False
        for line in f:
            if line.startswith('%FLAG RESIDUE_POINTER'):
                break
             if line.startswith('%FLAG RESIDUE_LABEL'):
                 residue_section = True
            if not residue_section or line.startswith('%F'):
                continue
            else:
                residue_names = line.split()
                for resi_name in residue_names:
                    if aa_code == 1:
                        resi_name = ThrLett_to_OneLett(resi_name)
                    resi_name = capitalize(resi_name) + str(resi_num + offset)
                    resi_map[resi_num + indexing - 1] = resi_name
                    resi_num += 1

    # If the file is a PDB file...

    else:
        for line in f:
            if not (line.startswith('ATOM') or line.startswith('HETATM')):
                continue
            resi_name = line[17:21].strip()
             resi_num = int(line[22:26].strip())
            if aa_code == 1:
                resi_name = ThrLett_to_OneLett(resi_name)
            resi_name = capitalize(resi_name) + str(resi_num + offset)
            resi_map[resi_num + indexing - 1] = resi_name

    f.close()
    return resi_map
```

setup_hbond_ptraj.py

```
#!/usr/bin/env python

'''
Set up ptraj input files to run a series of H-bond calculations.
'''

import sys, os
from optparse import OptionParser

if __name__ == '__main__':

    usage = """%prog [options]
This program sets up ptraj input files to perform a series of H-bond
analyses on a trajectory.  The trajectory is broken into segments of a
specified size.  Files written out include a ptraj input file for each
```

```
segment and a file "run_hbond_ptraj" that will sequentially execute
ptraj with each one.

The only required option is --mdcrd-file (-x), which specifies the
coordinate file.

Important notes:
* A file called "mask" can be created to include hydrogen-bonding
  residues other than the standard amino acids and water.  The lines in
  mask will be copied "as is" into each ptraj input file, so follow the
  format for specifying hydrogen bond donors and acceptors in the ptraj
  documentation."""

    # Parse command line options

    parser = OptionParser(usage = usage)

    parser.add_option('-x', '--mdcrd-file',
           dest = 'mdcrd_file',
           help = 'Coordinate file to use for H-bond analysis (REQUIRED).  [default:
%default]')
    parser.add_option('-p', '--prmtop-file',
           dest = 'prmtop_file',
           help = 'Amber parameter/topology file to use for H-bond analysis.  If None,
the name will be guessed by replacing the extension of the coordinate file name with
".prmtop"  [default: %default]')
    parser.add_option('-D', '--mdcrd-prmtop-dir',
           dest = 'mdcrd_prmtop_dir',
           help = 'The directory that contains the coordinate and prmtop file.  If None,
the names of the files will be used without modification.  [default: %default]')
    parser.add_option('-o', '--output-file-base',
           dest = 'output_file_base', default = 'segment',
           help = 'The first part of the name of the ptraj input files that will be
created.  The files will be named OUTPUT_FILE_BASEnn.in, where nn is the two-digit
segment number.  Once ptraj is run (with run_hbond_ptraj), the output files containing
the H-bond data will be named OUTPUT_FILE_BASEnn.out.  [default: %default]')
    parser.add_option('-B', '--hbond-data-dir',
           dest = 'hbond_data_dir', default = '.',
           help = 'The directory where the ptraj input files will be placed.  [default:
%default]')
    parser.add_option('-b', '--begin-frame', type = 'int',
           dest = 'begin_frame', default = 1,
           help = 'The number of the first frame to use in the H-bond analysis.
[default: %default]')
    parser.add_option('-e', '--end-frame', type = 'int',
           dest = 'end_frame', default = 10000,
           help = 'The number of the last frame to use in the H-bond analysis.  [default:
%default]')
    parser.add_option('-g', '--segment-size', type = 'int',
           dest = 'segment_size', default = 1000,
           help = 'The number of frames included in each segment of the trajectory.
[default: %default]')
    parser.add_option('-n', '--num-resi', type = 'int',
           dest = 'num_resi',
           help = 'The number of residues in the protein.  [default: 10000]')
    parser.add_option('-d', '--dist-cutoff', type = 'float',
           dest = 'dist_cutoff', default = 3.0,
           help = 'The distance cutoff that defines whether or not an interaction is an
H-bond.  [default: %default]')
    parser.add_option('-a', '--angle-cutoff', type = 'float',
           dest = 'angle_cutoff', default = 120.0,
           help = 'The angle cutoff that defines whether or not an interaction is an H-
bond.  [default: %default]')
    parser.add_option('-s', '--no-self', action = 'store_false',
           dest = 'self', default = True,
           help = 'Flag to turn off the inclusion of H-bonds between atoms within the
same residue.  [default: %default]')
    parser.add_option('-S', '--solvent', action = 'store_true',
           dest = 'solvent', default = False,
           help = 'Flag to include solvent-protein interactions in the analysis.
[default: %default]')
    parser.add_option('-m', '--mask-file',
           dest = 'mask_file', default = None,
           help = 'File that contains extra lines to include in the ptraj input files,
primarily to include masks for ligands.  [default: mask]')

    options, args = parser.parse_args()

    mdcrd_file = options.mdcrd_file
    prmtop_file = options.prmtop_file
    mdcrd_prmtop_dir = options.mdcrd_prmtop_dir
    output_file_base = options.output_file_base
    begin_frame = options.begin_frame
    end_frame = options.end_frame
    segment_size = options.segment_size
    num_resi = options.num_resi
    dist_cutoff = options.dist_cutoff
    angle_cutoff = options.angle_cutoff
    self = options.self
    solvent = options.solvent
    mask_file = options.mask_file
    hbond_data_dir = options.hbond_data_dir

    # Do error check for file names

    if not mdcrd_file:
        sys.exit('ERROR:  No coordinate file given (-x, --mdcrd-file option).  For help,
\n' + \
                 '  use the -h option.\n')
    if not prmtop_file:
        prmtop_file = os.path.splitext(options.mdcrd_file)[0] + '.prmtop'
        print 'Warning:  No prmtop file specified.  Will use ' + prmtop_file + '.'
    if mdcrd_prmtop_dir:
        mdcrd_file = os.path.join(mdcrd_prmtop_dir, mdcrd_file)
        prmtop_file = os.path.join(mdcrd_prmtop_dir, prmtop_file)

    if not os.path.exists(mdcrd_file):
        sys.exit('ERROR:  Coordinate file ' + mdcrd_file + '\n' + \
                 '  does not exist.')
    if not os.path.exists(prmtop_file):
        sys.exit('ERROR:  prmtop file ' + prmtop_file + '\n' + \
                 '  does not exist.')

    # Read through prmtop file to determine which residue types
    # are present and also get the residue numbers of the prolines

    prmtop_f = file(prmtop_file)

    prolines = []    # list of proline residue numbers
    resi_num = 1
    is_present = {}  # dict of residue types present in the file
    residue_section = False
    for line in prmtop_f:
        if line.startswith('%FLAG RESIDUE_POINTER'):
            break
        if line.startswith('%FLAG RESIDUE_LABEL'):
            residue_section = True
        if not residue_section or line.startswith('%F'):
            continue
        else:
            residue_names = line.split()
            for resi_name in residue_names:
                is_present[resi_name] = True
                if resi_name == 'PRO' and resi_num > 1:
                    prolines.append(resi_num)
                resi_num += 1

    resi_types = '''ALA ARG ASH ASN ASP CYM CYS CYX GLH GLN GLU GLY
                   HID HIE HIP ILE LEU LYN LYS MET PHE PRO SER THR
                   TRP TYR VAL'''.split()

    for resi_type in is_present:
        if resi_type not in resi_types:
            print 'Warning:  Unrecognized residue type ' + resi_type + ' is present in
system.  Atoms in\n' + \
                 '  this residue will not be included in H-bond analysis unless its\n' +
\
                 '  acceptors are specified in the mask file.'

    # Set non-None defaults for the options that need them

    if mask_file == None:
        mask_file = 'mask'
        mask_warning = ''
    else:
        mask_warning = 'Warning:  Could not open ' + mask_file + '.  Ignoring -m option.'
    extra_masks = ''
    try:
        mask_f = file(mask_file)
    except IOError:
        if mask_warning:
            print mask_warning
    else:
        for line in mask_f:
            extra_masks += line

    if num_resi == None:
        print 'Warning:  No num_resi option (-n, --num-resi) option specified.  Will
use\n' + \
              '  default value of 10000.  If your protein has fewer than 1000 residues\n'
+ \
              '  and there is solvent in the system, some of the solvent oxygen atoms\n'
+ \
              '  will be treated explicitly as donors.'
        num_resi = 10000

    # Create string of nonproline residues (- and , separated
    # like in normal ptraj input) based on proline list

    for resinum in range(2, num_resi + 1):
        if resinum in prolines:
            prolines = prolines[1:]
        else:
            prolines = [resinum - 1] + prolines + [num_resi + 1]
            break
    nonpro_str = ''
    for i in range(len(prolines) - 1):
        diff = prolines[i + 1] - prolines[i]
        if diff > 1:
            nonpro_str += str(prolines[i] + 1)
            if diff > 2:
               nonpro_str += '-' + str(prolines[i + 1] - 1)
            if i != len(prolines) - 2:
               nonpro_str += ','

    # Set strings of H-bond donor and acceptor atoms

    donor_mask = {}
    donor_mask['ASH'] = 'donor mask :ASH@OD1\n' + \
                       'donor mask :ASH@OD2'
    donor_mask['ASN'] = 'donor mask :ASN@OD1'
    donor_mask['ASP'] = 'donor mask :ASP@OD1\n' + \
                       'donor mask :ASP@OD2'
    donor_mask['CYM'] = 'donor mask :CYM@SG'
    donor_mask['GLH'] = 'donor mask :GLH@OE1\n' + \
                       'donor mask :GLH@OE2'
    donor_mask['GLN'] = 'donor mask :GLN@OE1'
    donor_mask['GLU'] = 'donor mask :GLU@OE1\n' + \
                       'donor mask :GLU@OE2'
    donor_mask['HID'] = 'donor mask :HID@NE2'
    donor_mask['HIE'] = 'donor mask :HIE@ND1'
    donor_mask['LYN'] = 'donor mask :LYN@NZ'
    donor_mask['SER'] = 'donor mask :SER@OG'
    donor_mask['THR'] = 'donor mask :THR@OG1'
    donor_mask['TYR'] = 'donor mask :TYR@OH'

    acceptor_mask = {}
    acceptor_mask['ARG'] = 'acceptor mask  :ARG@NH2 :ARG@HH21\n' + \
                         'acceptor mask  :ARG@NH2 :ARG@HH22\n' + \
                         'acceptor mask  :ARG@NH1 :ARG@HH11\n' + \
                         'acceptor mask  :ARG@NH1 :ARG@HH12\n' + \
                         'acceptor mask  :ARG@NE  :ARG@HE'
    acceptor_mask['ASH'] = 'acceptor mask  :ASH@OD2 :ASH@HD2'
    acceptor_mask['ASN'] = 'acceptor mask  :ASN@ND2 :ASN@HD21\n' + \
                         'acceptor mask  :ASN@ND2 :ASN@HD22'
    acceptor_mask['GLH'] = 'acceptor mask  :GLH@OE2 :GLH@HE2'
    acceptor_mask['GLN'] = 'acceptor mask  :GLN@NE2 :GLN@HE21\n' + \
                         'acceptor mask  :GLN@NE2 :GLN@HE22'
    acceptor_mask['HID'] = 'acceptor mask  :HID@ND1 :HID@HD1'
    acceptor_mask['HIE'] = 'acceptor mask  :HIE@NE2 :HIE@HE2'
    acceptor_mask['HIP'] = 'acceptor mask  :HIP@ND1 :HIP@HD1\n' + \
                         'acceptor mask  :HIP@NE2 :HIP@HE2'
    acceptor_mask['LYN'] = 'acceptor mask  :LYN@NZ  :LYN@HZ2\n' + \
                         'acceptor mask  :LYN@NZ  :LYN@HZ3'
    acceptor_mask['LYS'] = 'acceptor mask  :LYS@NZ  :LYS@HZ1\n' + \
                         'acceptor mask  :LYS@NZ  :LYS@HZ2\n' + \
                         'acceptor mask  :LYS@NZ  :LYS@HZ3'
    acceptor_mask['SER'] = 'acceptor mask  :SER@OG  :SER@HG'
    acceptor_mask['THR'] = 'acceptor mask  :THR@OG1 :THR@HG1'
    acceptor_mask['TRP'] = 'acceptor mask  :TRP@NE1 :TRP@HE1'
    acceptor_mask['TYR'] = 'acceptor mask  :TYR@OH  :TYR@HH'

    # Create the ptraj input files and run_all file

    try:
        run_file = os.path.join(hbond_data_dir, 'run_hbond_ptraj')
        run_f = file(run_file, 'w')
    except IOError:
        sys.exit('ERROR:  Could not open ' + run_file + '.\n')

    num_segments = (end_frame - begin_frame + 1) / segment_size
    remainder = (end_frame - begin_frame + 1) % segment_size
    if remainder:
       print 'Warning:  The number of frames is not a multiple of the segment size.\n' +
\
              '  Ignoring final ' + str(remainder) + ' frames.'
        end_frame -= remainder

    for segment_index in range(num_segments):

        # Write ptraj input file

       filename_base = output_file_base
       if segment_index < 9:
           filename_base += '0'
       filename_base += str(segment_index + 1)
        output = ''

       # Trajectory to read in

        output += 'trajin ' + mdcrd_file + ' ' + \
                   str(begin_frame + segment_index*segment_size) + ' ' + \
                   str(begin_frame + (segment_index+1)*segment_size - 1) + '\n'

       # List the donor atoms of sidechains

       output += '\n# List of potential H-bond donors\n'
       for resi_type in is_present:
           if donor_mask.has_key(resi_type):
               output += donor_mask[resi_type] + '\n'

       # List the acceptor atoms of sidechains

       output += '\n# List of potential H-bond acceptors\n'
       for resi_type in is_present:
           if acceptor_mask.has_key(resi_type):
               output += acceptor_mask[resi_type] + '\n'

       # List the backbone atoms

       output += '\n#-- Backbone donors and acceptors for ' + \
                              'this particular molecule\n' + \
                   '#   N-H for prolines do not exist so ' + \
                              'are not in the mask\n'
        output += 'donor mask :1-' + str(num_resi) + '@O\n' + \
                    'acceptor mask  :' + \
                    nonpro_str + \
                    '@N :2-' + str(num_resi) + '@H'

        output += '\n#-- Terminal residues have different atom names\n' + \
                   'donor mask @OXT\n' + \
                   'acceptor mask :1@N :1@H1\n' + \
                   'acceptor mask :1@N :1@H2\n' + \
                   'acceptor mask :1@N :1@H3\n'

        # List the extra lines from mask file

        output += '\n#-- Masks supplied by user\n' + extra_masks
        if not extra_masks:
            output += '#   None given\n'

        # hbond process line

        output += '\nhbond distance ' + str(dist_cutoff) + \
                    ' angle ' + str(angle_cutoff)
        if solvent:
            output += ' solventneighbor 6 solventdonor WAT O' + \
                     ' solventacceptor WAT O H1 solventacceptor WAT O H2'
        output += ' series hb_all out ' + filename_base + '.out'
        if self:
            output += ' includeself\n'
        else:
            output += '\n'

        # Output the ptraj file for the segment

        try:
            output_file = filename_base + '.in'
            output_file = os.path.join(hbond_data_dir, output_file)
            output_f = file(output_file, 'w')
        except IOError:
            print 'ERROR:  Could not open ' + output_file + '.\n'
            sys.exit('  Writing out ptraj input file to stdout:\n' + output)
        else:
            output_f.write(output + '\n')
            output_f.close()

        # Update runfile

        run_file_output = 'ptraj ' + prmtop_file + \
                         ' ' + output_file + '\n'
        run_f.write(run_file_output)

    run_f.close()
    os.system('chmod +x ' + run_file)
```

hbond_analysis_utils.py

```
#!/usr/bin/env python

"""
Michael Lerner's hbond analysis, modified by Steve Spronk

Right now, just handles pasting together ptraj output.
"""

import copy,pprint,os,sys
from scipy import sqrt
from string import ascii_letters
from tool_utils import *
```

```
class Atom:
    def __init__(self, atom_name = None, resi_name = None, resi_num = None):
        """
        Solvent atoms will have atom OW or HW and resi WAT.
        """
        self.atom_name = atom_name
        self.resi_name = resi_name
        self.resi_num = resi_num
    def __eq__(self, other):
        return self.atom_name == other.atom_name and \
               self.resi_name == other.resi_name and \
               self.resi_num == other.resi_num
    def __ne__(self, other):
        return not (self == other)

class HBond:
    """
    Class to provide a mechanism for handing data contained in the output
    from ptraj
    """

    # ---------------
    # Initializations
    # ---------------


    def __init__(self, line = None, segment_size = 1000, resi_map = None):
        '''
        Initialize ourself from a line that looks like this:
                DONOR         ACCEPTORH      ACCEPTOR
          atom# :res@atom   atom# :res@atom atom# :res@atom %occupied  distance
angle              lifetime maxocc
        |  2546 :160@OA23|  1018   :63@HG   1017   :63@OG  |  99.50  2.641 ( 0.10)  20.89
( 9.75)    100.0 ( 47.0)    147 |@@@*@@@@@|
        |  2545 :160@OA22|   705   :44@HH22  703   :44@NH2 |  98.51  2.756 (10.09)  17.97
(19.79)     99.0 (127.0)    126 |*@@@*@@@@|
        | solvent donor  |   127    :9@HD21  126    :9@ND2 |   2.00  3.193 ( 0.00)  46.59
( 0.01)      2.0 (  0.0)      1 |    .    |
        |  5612 :361@OG  |         solvent acceptor        |   2.00  2.915 ( 0.00)  11.31
( 0.00)      2.0 (  0.0)      1 |    .    |

        The numbers in parentheses are standard deviations.

        Here is a note from cheatham (http://amber.scripps.edu/Questions/mail/322.html)::

            The maxocc is the maximum number of consecutive frames that the
            interaction is found in the trajectory (i.e. 39 consecutive frames).

            The lifetime is the average time an interaction occurred...

            For example, assume that each space below represents 1ps and a star

            means it is occupied:

                     10        20         30        40        50
                 *****     *****     **********          *****|

            The occupancy would be 5 + 5 + 10 + 5 / 50 or 50%
            The maxocc would be 10
            The lifetime would be 5 + 5 + 10 + 5 / 4 = 6.25 ps (assuming 1 ps between
            frames; the time per frame can be specified on the hbond command line)

        Adding hbonds only works for some attributes (occupancy, distance, distance
        standard deviation, angle, angle standard deviation, and graph).

        But because we have split the trajectory into segments, the lifetime and maxocc
        are not truly a reflection of the H-bonds across the whole trajectory.
        Therefore the manipulation of lifetime and maxocc data are not implemented
        in the current version of hbond_analysis.
        '''

        # num_frames tells us how many frames have been added together.
        self.num_frames = segment_size
        if line is None:
            self.donor = Atom()
            self.acceptorh = Atom()
            self.acceptor = Atom()
            self.occ_pct = self.occ_num = self.dist = self.dist_stdev = self.angle =
self.angle_stdev = 0.0
            self.graph =  '          '
            return

        line = line.strip()
        try:
            leading_junk, donor, acceptor, stats, graph, trailing_junk = line.split('|')
        except ValueError:
            print "Could not hbond", line
            raise

        # Parse line:

        self.donor = self._ptraj_hbond_chunk_to_atom(donor, resi_map)
        self.acceptorh = self._ptraj_hbond_chunk_to_atom(' '.join(acceptor.split()[:2]),
resi_map)
        self.acceptor = self._ptraj_hbond_chunk_to_atom(' '.join(acceptor.split()[2:]),
resi_map)

        occ_pct,dist = stats.split('(')[0].strip().split()
        dist_stdev = stats.split('(')[1].split(')')[0].strip()
        angle = stats.split(')')[1].split('(')[0].strip()
        angle_stdev = stats.split('(')[2].split(')')[0].strip()

        # Make necessary type adjustments and calculations

        self.occ_pct,self.dist,self.dist_stdev,self.angle,self.angle_stdev = [float(i)
for i in occ_pct,dist,dist_stdev,angle,angle_stdev]
        self.graph = graph
        self.occ_num = int(round(self.occ_pct / 100.0 * self.num_frames))
        if self.occ_num < 2:
            self.dist_stdev = self.angle_stdev = 0.0
        self.straight_from_ptraj = True

    def _ptraj_hbond_chunk_to_atom(self, chunk, resi_map = None):
        ''' chunk is something like "  2546 :160@OA23  " '''
        if chunk.strip() in ('solvent donor', ''):
            return Atom(atom_name = 'OW', resi_name = 'Wat', resi_num = 999999)
        elif chunk.strip() == 'solvent acceptor':
            return Atom(atom_name = 'HW', resi_name = 'Wat', resi_num = 999999)
        else:
            resi_name = chunk.split(':')[1].split('@')[0].strip()
            if resi_map != None:
                resi_name = resi_map[int(resi_name)]
                try:
                    resi_num = int(resi_name)           # no aa code
                except ValueError:
                    if resi_name[1] in ascii_letters:   # 3-letter aa code
                        resi_num = int(resi_name[3:])
                    else:                               # 1-letter aa code
                        resi_num = int(resi_name[1:])
            else:
                resi_num = int(resi_name)
            atom_name = chunk.split(':')[1].split('@')[1].strip()
            return Atom(atom_name, resi_name, resi_num)

    def init_from_atomstr(self, s, segment_size = 1000):
        '''
        atomstr looks like what is returned by self._atom_str:
            102 NH1--HH11 ... O     88
          Tyr71    OH--HH ... OG1   Asp228
        '''
        a_resi, a_atom, ah_atom, dots, d_atom, d_resi = s.replace('--',' ').split()

        if a_resi == 'Wat':
            a_resi_num = 999999
        else:
            try:
                a_resi_num = int(a_resi)        # no aa code
            except ValueError:
                if a_resi[1] in ascii_letters:  # 3-letter aa code
                    a_resi_num = int(a_resi[3:])
                else:                           # 1-letter aa code
                    a_resi_num = int(a_resi[1:])

        if d_resi == 'Wat':                # Same for donor atom
            d_resi_num = 999999
        else:
            try:
                d_resi_num = int(d_resi)
            except ValueError:
                if d_resi[1] in ascii_letters:
                    d_resi_num = int(d_resi[3:])
                else:
                    d_resi_num = int(d_resi[1:])

        self.donor     = Atom( d_atom, d_resi, d_resi_num)
        self.acceptor  = Atom( a_atom, a_resi, a_resi_num)
        self.acceptorh = Atom(ah_atom, a_resi, a_resi_num) # H is always in same residue
as heavy atom it's bonded to

        self.num_frames = segment_size
        self.occ_num = 0
        self.occ_pct = self.dist = self.dist_stdev = self.angle = self.angle_stdev = 0.0
        self.graph =  '          '
        self.straight_from_ptraj = True

    def init_from_str(self, s):
        """
        str looks like what is output by __str__ :
         Lys142   NZ--HZ3 ... OE1  Glu134  27.80( 2500) |--... .*-.|oo---x- - |
        or what is output by _attr_str:
         Lys142   NZ--HZ3 ... OE1  Glu134  27.80( 2500) 2.850(0.17) 29.14(15.66) |--...
.*-.|oo---x- - |
        """
        atom_str_len = 34
        hbond_name = s[:atom_str_len].strip()
        hbond_attr = s[atom_str_len:].strip()

        # Take care of Atoms first

        self.init_from_atomstr(hbond_name)

        # Then take care of attributes

        try:
            attr_list = hbond_attr.split(')')

            # Attributes from __str__
            self.occ_pct = float(attr_list[0].split('(')[0])
            self.num_frames = int(attr_list[0].split('(')[1])
            self.graph = attr_list[-1].strip()[1:-1]  # The [1:-1] takes care of leading
and trailing '|'

            # If present, attributes from _attr_str
            attr_list = attr_list[1:-1]
            if attr_list != []:
                self.dist = float(attr_list[0].split('(')[0])
                self.dist_stdev = float(attr_list[0].split('(')[1])
                self.angle = float(attr_list[1].split('(')[0])
                self.angle_stdev = float(attr_list[1].split('(')[1])
        except:
            print "String could not be converted to hbond:", s
            raise

        self.occ_num = int(round(self.occ_pct / 100.0 * self.num_frames))
        self.straight_from_ptraj = False

    # ---------------
    # Representations
    # ---------------
    def __str__(self):
        return self._atom_str() + ' ' + self._occ_graph_str()

    def _atom_str(self):
        """
        Returns the atoms identifying the Hbond as a formatted string.
        Examples:
            102 NH1--HH11 ... O    88
          Tyr71    OH--HH ... OG1  Asp228
        """
        spaces = (7 - len(self.acceptor.resi_name)) * ' '
        bond_string = spaces + self.acceptor.resi_name
        acceptor_str = "%s--%s"%(self.acceptor.atom_name,
                                 self.acceptorh.atom_name)
        spaces = (10 - len(acceptor_str)) * ' '
        bond_string += spaces + acceptor_str + " ... "
        spaces = (5 - len(self.donor.atom_name)) * ' '
        bond_string += self.donor.atom_name + spaces
        spaces = (7 - len(self.donor.resi_name)) * ' '
        return bond_string + self.donor.resi_name + spaces

    def _attr_str(self):
        """
        Returns the attributes in a formatted string.
        """
        return "%6.2f(%5s)%6.3f(%4.2f)%6.2f(%5.2f) |%s|"\
                %(self.occ_pct, self.num_frames, self.dist, self.dist_stdev,
                  self.angle, self.angle_stdev, self.graph,)

    def _occ_graph_str(self):
        """
        Returns the occupancy, count, and graph in a formatted string.
        """
        return "%6.2f(%5s) |%s|"%(self.occ_pct, self.num_frames, self.graph)

    __repr__ = __str__

    # ----------
    # Operations
    # ----------

    def __add__(self,other):
        """
        Combines the statistics of two hbonds.  The new number of frames, number of
        occupied frames, occupancy percentage, distance, angle, distance standard
        deviation, angle standard devation, and graph are all accurately calculated.

        A note on the standard deviation calculations:  ptraj calculates sigma as the
        standard deviation (which has N in the denominator).  This is not strictly
        correct, as this formula only holds true if we know all of the data.  However,
        we know that our data only contains a sampling from the actual ensemble, so
        it we should use the estimated population standard deviation (S) of the
        statistics, which has N-1 in the denominator of the calculation.
        """
        if type(self) != type(other):
            raise Exception('Cannot add hbond to non-hbond %s object:
%s'%(type(other),other))

        if self._atom_str() != other._atom_str():
            raise Exception('Can only add hbonds with the same donors and acceptors\n' \
                            '%s != %s'%(self._atom_str(),other._atom_str()))

        result = HBond()

        result.donor = Atom(self.donor.atom_name, self.donor.resi_name,
self.donor.resi_num)
        result.acceptor = Atom(self.acceptor.atom_name, self.acceptor.resi_name,
self.acceptor.resi_num)
        result.acceptorh = Atom(self.acceptorh.atom_name, self.acceptor.resi_name,
self.acceptor.resi_num)

        result.num_frames = self.num_frames + other.num_frames
        sep = '|'
        result.graph = self.graph + sep + other.graph
        result.occ_num = self.occ_num + other.occ_num
        result.occ_pct = result.occ_num * 100.0 / result.num_frames
        result.straight_from_ptraj = False

        if result.occ_num > 0:

            result.dist  = (self.occ_num * self.dist  + other.occ_num * other.dist ) /
result.occ_num
            result.angle = (self.occ_num * self.angle + other.occ_num * other.angle) /
result.occ_num

            # It's relatively complicated to calculate the new standard deviation.  See
my Notebook 3,
            # pp. 72-4 for the derivation.  We must make a distinction on whether or not
the data is
            # straight from the ptraj files, because when we are looking at the data from
ptraj
            # (straight_from_ptraj = True) the std. dev. is actually sigma as opposed to
S, the estimated
            # standard deviation of the population.  In practice, these values are close
(for relatively
            # large N), but I want to be precise with my statistics.

            if result.occ_num == 1:
                result.dist_stdev = result.angle_stdev = 0.0

            else:
                dist_sumsq = angle_sumsq = 0.0
                if self.straight_from_ptraj:
                    dist_sumsq  += self.dist_stdev      * self.dist_stdev      *
self.occ_num + \
                                   self.dist            * self.dist            *
self.occ_num
                    angle_sumsq += self.angle_stdev     * self.angle_stdev     *
self.occ_num + \
                                   self.angle           * self.angle           *
self.occ_num
                else:
                    dist_sumsq  += self.dist_stdev      * self.dist_stdev      *
(self.occ_num - 1) + \
                                   self.dist            * self.dist            *
self.occ_num
                    angle_sumsq += self.angle_stdev     * self.angle_stdev     *
(self.occ_num - 1) + \
                                   self.angle           * self.angle           *
self.occ_num
                if other.straight_from_ptraj:
                    dist_sumsq  += other.dist_stdev     * other.dist_stdev     *
other.occ_num + \
                                   other.dist           * other.dist           *
other.occ_num
                    angle_sumsq += other.angle_stdev    * other.angle_stdev    *
other.occ_num + \
                                   other.angle          * other.angle          *
other.occ_num
                else:
                    dist_sumsq  += other.dist_stdev     * other.dist_stdev     *
(other.occ_num - 1) + \
                                   other.dist           * other.dist           *
other.occ_num
                    angle_sumsq += other.angle_stdev    * other.angle_stdev    *
(other.occ_num - 1) + \
                                   other.angle          * other.angle          *
other.occ_num

                result.dist_stdev     = sqrt((dist_sumsq  - result.occ_num*result.dist
*result.dist    ) / (result.occ_num - 1))
                result.angle_stdev    = sqrt((angle_sumsq - result.occ_num*result.angle
*result.angle   ) / (result.occ_num - 1))

        #else:
```

```
        #    result.dist = result.dist_stdev = result.angle = result.angle_stdev = 0.0

        return result

    def compress_graph(self):
        """
        Compresses the graph of a trajectory into one half the size.
        Each pair of characters is replaced by a single character
        that is representative of the percentage of occupancy for
        the union of the two segments.  Unfortunately, the actual
        occupancy percentage of the union can not be absolutely
        determined from the two symbols of the graph, so the new
        graph may not be precise.  See my Notebook 3, pp. 78-79
        for a detailed analysis of how I determined how two
        symbols should be combined.
        """
        graph_sections = self.graph.split('|')
        new_graph = ''
        for graph_num in range(len(graph_sections)):
            for i in range(0, 10, 2):
                pair = graph_sections[graph_num][i:i+2]
                if pair[0] == pair[1]:
                    new_graph += pair[0]

                elif pair == ' .' or pair == '. ':
                    new_graph += '.'
                 elif pair == ' -' or pair == '- ':
                     new_graph += '.'
                 elif pair == ' o' or pair == 'o ':
                     new_graph += '-'
                 elif pair == ' x' or pair == 'x ':
                     new_graph += '-'
                 elif pair == ' *' or pair == '* ':
                     new_graph += 'o'
                 elif pair == ' @' or pair == '@ ':
                     new_graph += 'o'
                 elif pair == '.-' or pair == '-.':
                     new_graph += '-'
                 elif pair == '.o' or pair == 'o.':
                     new_graph += '-'
                 elif pair == '.x' or pair == 'x.':
                     new_graph += 'o'
                 elif pair == '.*' or pair == '*.':
                     new_graph += 'o'
                 elif pair == '.@' or pair == '@.':
                     new_graph += 'o'
                 elif pair == '-o' or pair == 'o-':
                     new_graph += 'o'
                 elif pair == '-x' or pair == 'x-':
                     new_graph += 'o'
                 elif pair == '-*' or pair == '*-':
                     new_graph += 'o'
                 elif pair == '-@' or pair == '@-':
                     new_graph += 'x'
                 elif pair == 'ox' or pair == 'xo':
                     new_graph += 'o'
                 elif pair == 'o*' or pair == '*o':
                     new_graph += 'x'
                 elif pair == 'o@' or pair == '@o':
                     new_graph += 'x'
                 elif pair == 'x*' or pair == '*x':
                     new_graph += 'x'
                 elif pair == 'x@' or pair == '@x':
                     new_graph += '*'
                elif pair == '*@' or pair == '@*':
                     new_graph += '*'

            if graph_num % 2 == 1:
                new_graph += '|'

        if new_graph[-1] == '|':
            self.graph = new_graph[:-1]
        else:
           self.graph = new_graph

    # ------ End class HBond ----

def hbond_lines(lines):
    reading = False
    for line in lines:
        if line.strip() == '  atom# :res@atom   atom# :res@atom atom# :res@atom %occupied
distance       angle              lifetime maxocc'.strip():
            reading = True
        if not reading or line.strip().startswith('atom') or not line.replace('-
','').strip():
            continue
        yield line
def hbonds_from_ptraj(f, segment_size = 1000, resi_map = None):
    return [HBond(line, segment_size, resi_map) for line in hbond_lines(f)]

def is_resinum_of_interest(hbond, criteria = ['all']):
    """
    Tells us if a hbond has a residue number among those we want to view
    """
    if 'all' in criteria:
        return True
    if hbond.donor.resi_num in criteria or hbond.acceptor.resi_num in criteria:
       return True
    else:
       return False

def is_atom_of_interest(hbond, criteria = ['all']):
    """
    Tells us if an hbond has an atom type among those we want to view
    """
    if 'all' in criteria:
       return True
    if 'protein_only' in criteria:
       if hbond.donor.atom_name == 'OW' or hbond.acceptor.atom_name == 'OW':
           return False
       else:
           return True
    if 'bb_only' in criteria:
       if hbond.donor.atom_name == 'O' and hbond.acceptor.atom_name == 'N':
           return True
    if 'not_bb' in criteria:
       if hbond.donor.atom_name != 'O' or hbond.acceptor.atom_name != 'N':
           return True
    if hbond.donor.atom_name in criteria or \
       hbond.acceptor.atom_name in criteria or \
       hbond.acceptorh.atom_name in criteria:
       return True
    else:
       return False

def combine_hbonds(hbond_files, segment_size = 1000,
               resi_map = None, output_file = None,
               resi_criteria = ['all'], atom_criteria = ['all'],
               occ_thresh = 0.0, occ_graph_only = False,
               hbond_data_dir = None):
    """
    Reads through a set of files that have been output by ptraj and compiles
    all the data.

    hbond_files:  the hbond_files output from ptraj to be combined.
    segment_size:  the number of frames included in each segment of the
       trajectory. (default: 1000)
    resi_map:  a dictionary mapping the name of each residue onto the residue
       number.  If 'None,' the residue name will simply be the number.
       (default: None)
    output_file:  the name of the output file.  If None, the results will be
        written to stdout. (default: None)
    resi_criteria:  a list containing residue number criteria to include in the
       output. (default: ['all'])
    atom_criteria:  a list containing atom name criteria to include in the
       output. (default: ['all'])
    occ_thresh:  the minimum occupancy threshold that the hbonds must have
        to be reported. (default: 0.0)
    occ_graph_only:  if True, only the atom string, occupancy, and graph of
        each hbond will be written to output. (default: False)
    hbond_data_dir:  the directory that contains the hbond data files.  If
        'None,' the file names will be used without modification, and the
        output will be written to the current directory. (default: None)
    """

    # Do error checking of file names

    files_to_remove = []
    for each_file in hbond_files:
        if hbond_data_dir != None:
            full_file = os.path.join(hbond_data_dir, each_file)
        else:
            full_file = each_file
        if not os.path.exists(full_file):
            print 'Warning:  File ' + full_file + ' does not exist.\n' + \
                  '  Will be ignored.'
            files_to_remove.append(each_file)
    for each_file in files_to_remove:
        hbond_files.remove(each_file)
    if len(hbond_files) == 0:
        sys.exit('ERROR:  No input files provided.\n')

    # Create list of hbonds in each file, and a master hbond dict

    hbonds_from_file = {}  # {filename: list of hbond objects}
    combined_hbonds = {}   # {hbond string: hbond object}

    for each_file in hbond_files:
        if hbond_data_dir != None:
            hbond_file = os.path.join(hbond_data_dir, each_file)
        else:
            hbond_file = each_file
        try:
            hbond_f = file(hbond_file)
        except:
            sys.exit('ERROR:  Could not open ' + hbond_file + '.\n')
        hbonds_from_file[each_file] = hbonds_from_ptraj(hbond_f, segment_size, resi_map)
        for hbond in hbonds_from_file[each_file]:
            combined_hbonds[hbond._atom_str()] = None

    # Run through the master hbond dict, and find out the missing hbonds
    # in each file.  If any are missing, create an hbond with no occupancy.

    for each_file in hbond_files:
        for hbond_str in combined_hbonds:
            found = False
            for hbond in hbonds_from_file[each_file]:
                if hbond._atom_str() == hbond_str:
                    found = True
                    break
            if not found:
                hbond = HBond()
                hbond.init_from_atomstr(hbond_str, segment_size)
                hbonds_from_file[each_file].append(hbond)

    # Do the addition of the hbonds from each file to create the
    # final combined hbond object.

    for hbond in hbonds_from_file[hbond_files[0]]:
        combined_hbonds[hbond._atom_str()] = hbond
    for each_file in hbond_files[1:]:
        for hbond in hbonds_from_file[each_file]:
            combined_hbonds[hbond._atom_str()] = combined_hbonds[hbond._atom_str()] +
hbond

    # Write output to file or stdout

    output = []
    for hbond in combined_hbonds.values():
        if is_resinum_of_interest(hbond, resi_criteria) and \
          is_atom_of_interest(hbond, atom_criteria) and \
          hbond.occ_pct > occ_thresh:
            if not occ_graph_only:
               output.append((hbond.occ_pct, hbond._atom_str() + ' ' +
hbond._attr_str()))
           else:
               output.append((hbond.occ_pct, str(hbond)))
    output.sort()
    output.reverse()
    output = [o[1] for o in output]
    output_str = '\n'.join(output)

    if hbond_data_dir == None:
       output_dir = '.'
    else:
       output_dir = hbond_data_dir

    if output_file == None:
       print output_str
    else:
       try:
           output_file = os.path.join(output_dir, output_file)
           output_f = file(output_file, 'w')
       except IOError:
           print 'Warning:  Could not open ' + output_file + '.\n'
           print output_str
       else:
           output_f.write(output_str + '\n')
           output_f.close()

def subset_hbonds(hbond_file, output_file = None,
               resi_criteria = ['all'], atom_criteria = ['all'],
               occ_thresh = 0.0, occ_graph_only = False,
               sort = 'occ_pct', compress = False,
               hbond_data_dir = None):
    """
    Following combination of hbonds by combine_hbond(), this function can be
    used to write to stdout or a file only a subset of all the data present.

    hbond_file:  the hbond file with data to be analyzed.
    output_file:  the name of the output file.  If None, the results will be
        written to stdout. (default: None)
    resi_criteria:  a list containing residue number criteria to include in the
        output. (default: ['all'])
    atom_criteria:  a list containing atom name criteria to include in the
        output. (default: ['all'])
    occ_thresh:  the minimum occupancy threshold that the hbonds must have
        to be reported. (default: 0.0)
    occ_graph_only:  if True, only the atom string, occupancy, and graph of
        each hbond will be written to output. (default: False)
    sort:  one of 'occ_pct', 'donor', 'acceptor', 'dist', or 'angle' that
        indicates how to sort the output. (default: occ_pct)
    compress:  if True, the graphs will be compressed by compress_graph().
        (default: False)
    hbond_data_dir:  the directory that contains the hbond data files.  If
        'None,' the file names will be used without modification, and the
        output will be written to the current directory. (default: None)
    """
    # Do error checking of file names.

    if not hbond_file:
       sys.exit('ERROR:  No input file provided.\n')
    if type(hbond_file) is type([]):
       if len(hbond_file) > 1:
           print 'Warning:  More than 1 input file provided.\n' + \
                 '  Will only use first one: ' + hbond_file[0]
       hbond_file = hbond_file[0]
    if hbond_data_dir != None:
        full_file = os.path.join(hbond_data_dir, hbond_file)
    else:
        full_file = hbond_file
    try:
```

```
    hbond_f = file(full_file)
except IOError:
    sys.exit('ERROR:  Could not open ' + full_file + '.\n')

# Create list of hbonds in the input file, check to see if they
# satisfy the necessary criteria for output.

hbond_list = []
for line in hbond_f:
    hbond = HBond()
    hbond.init_from_str(line)
    hbond_list.append(hbond)

output = []
for hbond in hbond_list:
    if is_resinum_of_interest(hbond, resi_criteria) and \
       is_atom_of_interest(hbond, atom_criteria) and \
       hbond.occ_pct > occ_thresh:
        if compress:
            hbond.compress_graph()

        if occ_graph_only:
            hbond_str = str(hbond)
        else:
            hbond_str = hbond._atom_str() + ' ' + hbond._attr_str()

        if sort not in 'occ_pct acceptor donor dist angle'.split():
            print 'Warning:  Unknown sorting method: ' + sort + '.\n'  + \
                  '  Will sort by occupancy percentage.'
            sort = 'occ_pct'

        if sort == 'occ_pct':
            output.append((hbond.occ_pct,
                          hbond.acceptor.resi_num,
                          hbond_str))
        elif sort == 'acceptor':
            output.append((hbond.acceptor.resi_num,
                          hbond.acceptor.atom_name,
                           hbond.donor.resi_num,
                          hbond.donor.atom_name,
                           hbond_str))
        elif sort == 'donor':
            output.append((hbond.donor.resi_num,
                          hbond.donor.atom_name,
                           hbond.acceptor.resi_num,
                          hbond.acceptor.atom_name,
                           hbond_str))
        elif sort == 'dist':
            output.append((hbond.dist, hbond_str))
        else:                          # sort must be 'angle'
            output.append((hbond.angle, hbond_str))

# Write output

output.sort()
if sort == 'occ_pct':
    output.reverse()
output = [o[-1] for o in output]
output_str = '\n'.join(output)

if hbond_data_dir == None:
    output_dir = '.'
else:
    output_dir = hbond_data_dir

if output_file == None:
    print output_str
else:
    try:
        output_file = os.path.join(output_dir, output_file)
        output_f = file(output_file, 'w')
    except IOError:
        print 'Warning:  Could not open ' + output_file + '.\n'
```

```
            print output_str
        else:
            output_f.write(output_str + '\n')
            output_f.close()

def compare_hbonds(hbond_files, identifiers = [], output_file = None,
                resi_criteria = ['all'], atom_criteria = ['all'],
                occ_thresh = 0.0, occ_graph_only = False,
                sort = 'occ_diff', compress = False,
                hbond_data_dir = None):
    """
    Following combination of hbonds by combine_hbond() for distinct
    trajectories, this function can be used to present the data as a
    side-by-side comparison of hbond occupancies.

    hbond_files:  the hbond files with data to be analyzed.
    identifiers:  the list of names associated with each hbond_file.  If
        the list is empty, each file will simply be assigned a number.
        (default: [])
    output_file:  the name of the output file.  If None, the results will be
        written to stdout. (default: None)
    resi_criteria:  a list containing residue number criteria to include in the
        output. (default: ['all'])
    atom_criteria:  a list containing atom name criteria to include in the
        output. (default: ['all'])
    occ_thresh:  the minimum occupancy threshold that the hbonds must have
        to be reported. (default: 0.0)
    occ_graph_only:  if True, only the atom string, occupancy, and graph of
        each hbond will be written to output. (default: False)
    sort:  one of 'occ_diff', 'occ_pct', 'donor', or 'acceptor' that
        indicates how to sort the output. (default: occ_diff)
    compress:  if True, the graphs will be compressed by compress_graph().
        (default: False)
    hbond_data_dir:  the directory that contains the hbond data files.  If
        'None,' the file names will be used without modification, and the
        output will be written to the current directory. (default: None)
    """
    # Set up identifier strings

    for i in range(len(hbond_files)):
       if i >= len(identifiers):
           identifiers.append(str(i + 1))
    max_id_length = max(len(id) for id in identifiers)
    for i in range(len(identifiers)):
       num_spaces = max_id_length - len(identifiers[i])
       identifiers[i] = num_spaces * ' ' + identifiers[i]

    # Do error checking on file names

    files_to_remove = []
    for each_file in hbond_files:
       if hbond_data_dir != None:
           full_file = os.path.join(hbond_data_dir, each_file)
       else:
           full_file = each_file
       if not os.path.exists(full_file):
           print 'Warning:  File ' + full_file + ' does not exist.\n' + \
                 '  Will be ignored.'
           files_to_remove.append(each_file)
    for each_file in files_to_remove:
       i = hbond_files.index(each_file)
       identifiers.remove(identifiers[i])
       hbond_files.remove(each_file)
    if len(hbond_files) == 0:
       sys.exit('ERROR:  No input files provided.\n')

    if hbond_data_dir != None:
       for i in range(len(hbond_files)):
           hbond_files[i] = os.path.join(hbond_data_dir, hbond_files[i])

    # Create dictionaries for each file indicating their hbonds

    hb_dict_list = []     # One dictionary per hbond input file
```

```
combined_hbonds = {}  # {hbond_string: None} just keeps cumulative track
for each_file in hbond_files:
    hb_dict = {}      # {hbond_string: hbond object}
    for line in file(each_file):
        hbond = HBond()
        hbond.init_from_str(line)
        if is_resinum_of_interest(hbond, resi_criteria) and \
           is_atom_of_interest(hbond, atom_criteria):
           if compress:
               hbond.compress_graph()
           hb_dict[hbond._atom_str()] = hbond
           combined_hbonds[hbond._atom_str()] = None
    hb_dict_list.append(hb_dict)

# Run through the master list of all hbonds.  If a given
# dictionary doesn't have an entry for one, create one with
# zero occupancy.

for hb_dict in hb_dict_list:
    for hbond_str in combined_hbonds:
        found = False
        for hbond_str_dict in hb_dict:
            if hbond_str_dict == hbond_str:
                found = True
                break
        if not found:
            hbond = HBond()
            hbond.init_from_atomstr(hbond_str)
            hb_dict[hbond_str] = hbond

# Compile and sort relevant data

if sort not in 'occ_diff occ_pct donor acceptor'.split():
    print 'Warning:  Unknown sorting method: ' + sort + '.\n' + \
          '  Will use occ_diff to sort.'
    sort = 'occ_diff'

output = []
for hbond_str in combined_hbonds:
    hb_list = [ hb_dict[hbond_str] for hb_dict in hb_dict_list ]

    max_occ = max(hbond.occ_pct for hbond in hb_list)
    min_occ = min(hbond.occ_pct for hbond in hb_list)
    occ_diff = max_occ - min_occ

    if sort == 'occ_diff' and occ_diff > occ_thresh:
        output.append((occ_diff,
                      hb_list[0].acceptor.resi_num,
                      hb_list))
    elif sort == 'occ_pct' and max_occ > occ_thresh:
        output.append((max_occ,
                      hb_list[0].acceptor.resi_num,
                      hb_list))
    elif sort == 'donor' and occ_diff > occ_thresh:
        output.append((hb_list[0].donor.resi_num,
                      hb_list[0].donor.atom_name,
                      hb_list[0].acceptor.resi_num,
                      hb_list[0].acceptor.atom_name,
                      hb_list))
    elif sort == 'acceptor' and occ_diff > occ_thresh:
        output.append((hb_list[0].acceptor.resi_num,
                      hb_list[0].acceptor.atom_name,
                      hb_list[0].donor.resi_num,
                      hb_list[0].donor.atom_name,
                      hb_list))

output.sort()
if sort == 'occ_diff' or sort == 'occ_pct':
    output.reverse()
output = [o[-1] for o in output]

# Write output
```

```
    output_str = ''
    for each_hbond in output:
       for i in range(len(each_hbond)):
           hbond = each_hbond[i]
            if occ_graph_only:
               output_str += identifiers[i] + ': ' + str(hbond) + '\n'
            else:
               output_str += identifiers[i] + ': ' + \
                              hbond._atom_str() + ' ' + hbond._attr_str() + '\n'
        output_str += '\n'

    if hbond_data_dir == None:
       output_dir = '.'
    else:
       output_dir = hbond_data_dir

    if output_file == None:
        print output_str[:-2]  # Removes the last 2 newlines
    else:
        try:
            output_file = os.path.join(output_dir, output_file)
            output_f = file(output_file, 'w')
        except IOError:
            print 'Warning:  Could not open ' + output_file + '.\n'
            print output_str
        else:
            output_f.write(output_str[:-1])
            output_f.close()
```

subset_hbonds.py

```
#!/usr/bin/env python

'''
Set up ptraj input files to run a series of H-bond calculations.
'''

import sys, os
from optparse import OptionParser
from tool_utils import parse_residue_list, parse_atom_list
from hbond_analysis_utils import subset_hbonds

if __name__ == '__main__':

    usage = """%prog FILE1 [options]
FILE1 is a file created by combine_hbonds.py that contains a dataset of
H-bonds.  Only a subset of all the data is presented according to the
criteria presented by the user.  The data will also be sorted according
to the metric specified by the user.

Important notes:
* The resi_criteria option takes a comma-separated string containing any
  or all of the following:
        - individual residue numbers
        - a range of numbers, separated by a '-'
        - strings associated with valid residue lists in the
            standard file 'residue_lists'
* The atom_criteria option takes a comma-separated string containing
  the atom names to report.  In addition, the string can contain any
  of these strings:
        - 'bb_only': only H-bonds between two backbone atoms
        - 'not_bb': no H-bonds between two backbone atoms
        - 'protein_only': no H-bonds involving water
* Note that ptraj uses a definition of H-bond donor and acceptor that is
  opposite of the normal convention.  This program follows the
  definitions of ptraj, in which the acceptor is the atom covalently
  bonded to the hydrogen atom."""
```

```
    # Parse command line options

    parser = OptionParser(usage = usage)

    parser.add_option('-o', '--output-file',
           dest = 'output_file',
           help = 'The name of the output file.  If None, the results will be written to
stdout.  [default: %default]')
    parser.add_option('-B', '--hbond-data-dir',
           dest = 'hbond_data_dir',
           help = 'The directory that contains the H-bond data files.  If None, the file
names will be used without modification, and output will be written to the current
directory.  [default: %default]')
    parser.add_option('-s', '--sort',
           dest = 'sort', default = 'occ_pct',
           choices = ['occ_pct', 'donor', 'acceptor', 'dist', 'angle'],
           help = 'The quantity used to sort the results.  Must be one of "occ_pct"
(occupancy percentage), "donor", "acceptor", "dist", or "angle".   [default: %default]')
    parser.add_option('-c', '--compress', action = 'store_true',
           dest = 'compress', default = False,
           help = 'Flag to compress the H-bond graph.  [default: %default]')
    parser.add_option('-y', '--occ-graph-only', action = 'store_true',
           dest = 'occ_graph_only', default = False,
           help = 'Flag to report only the occupancy and graph data (no distance or angle
data).  [default: %default]')
    parser.add_option('-R', '--resi-criteria',
           dest = 'resi_criteria', default = 'all',
           help = 'A comma- and dash-separated list of residue numbers to include in the
analysis.  [default: %default]')
    parser.add_option('-A', '--atom-criteria',
           dest = 'atom_criteria', default = 'all',
           help = 'A comma-separated list of atom names to include in the analysis.
[default: %default]')
    parser.add_option('-O', '--occ-thresh', type = 'float',
           dest = 'occ_thresh', default = 0.0,
           help = 'The minimum occupancy threshold that the H-bonds must have to be
reported.  [default: %default]')

    options, hbond_file = parser.parse_args()

    # Process options

    resi_criteria = parse_residue_list(options.resi_criteria)
    if not resi_criteria:
        sys.exit('ERROR:  No residues selected in residue string.\n')
    atom_criteria = parse_atom_list(options.atom_criteria)

    # Perform function

    subset_hbonds(hbond_file = hbond_file,
       output_file = options.output_file,
       resi_criteria = resi_criteria,
       atom_criteria = atom_criteria,
       occ_thresh = options.occ_thresh,
       occ_graph_only = options.occ_graph_only,
       sort = options.sort,
       compress = options.compress,
       hbond_data_dir = options.hbond_data_dir  )
```