Improving the Performance of ProMOL by Integrating Communal Database Access

Alexander Grier

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact rit scholarworks@rit.edu.
Improving the Performance of ProMOL by Integrating Communal Database Access

by

Alexander Grier

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Bioinformatics

Thomas H. Gosnell School of Life Sciences

College of Science

Rochester Institute of Technology

Rochester, NY

August 18, 2016
Thesis Committee Members

Paul A. Craig
School of Chemistry & Material Science
Rochester Institute of Technology

Michael V. Osier
Thomas H. Gosnell School of Life Sciences
Rochester Institute of Technology

Gregory A. Babbitt
Thomas H. Gosnell School of Life Sciences
Rochester Institute of Technology
Abstract

The ability of structural genomics consortia to determine protein structures more quickly than the proteins can be functionally annotated by conventional experimental means has generated a significant bottleneck that inhibits the progress of certain areas of biological and medical research. This issue is currently being addressed in part by the development of computational resources, which can suggest functions for proteins of interest based on their structure. ProMOL is such a resource, and the method that it employs is based on detecting, in structures of unknown function, motifs of known function from a template library. Access to ProMOL results has been limited, due to a number of factors including the computational requirements for generating novel results and the format in which results have been stored. In this project, a remote results database that can be shared by the ProMOL user community has been implemented and integrated into the ProMOL client program. The introduction of database features addresses several of the drawbacks ProMOL previously had, especially related to efficiency, redundant computations, and the ability to maximize the potential utility of accumulated analysis results. Using the database has no effect on performance in the worst case scenario, and in the best case scenario it decreases runtime by ≥99%. Analyses of accumulated results in the database facilitate better quantification of the accuracy and performance of ProMOL, and suggest ways to improve the program. Finally, the database constitutes a novel resource with a variety of potential avenues of investigation that have not been possible previously.
Introduction

Structural motifs corresponding to enzyme active sites are often highly conserved within evolutionarily related proteins, and can be very similar between unrelated proteins with similar function. In cases where functional conservation is observed in association with these motifs, such similarities can be the basis for inferring function from structure, and computational structural analysis using this approach has become an active area of bioinformatics research in response to an increasing need for high throughput analysis of the tremendous quantities of data being generated by structural genomics consortia. This area of research is concerned with predicting the function of proteins with known structure and unknown function, and with contributing to a better understanding of the structural basis of enzyme activity.\(^1\) Numerous software applications have been developed for these purposes.\(^2\) One such application is ProMOL,\(^3\) a molecular visualization and analysis tool which functions as a plugin for the widely used molecular graphics system PyMOL.\(^4\) ProMOL is a template-based motif identification program being developed collaboratively and distributed freely as open source. The availability, flexibility, and open source design philosophy of ProMOL are conducive to the implementation of a crowdsourcing usage model via the integration of a communal results database. Such an approach can be used to address several of the most significant technical and theoretical weaknesses of the software while extending its capabilities and potential applications substantially. This discussion constitutes a brief review of the biological basis and design considerations of computational methods of protein structure analysis; a description of ProMOL in its original form; an explanation of the rationale and design of improvements based on the addition of database functions to the software and usage model; a summary of the effects of these additions; and several suggestions for promising future work utilizing these tools.
Broadly speaking, a protein motif is a recurrent element at the level of sequence or structure.\(^5\) The recurrence of these elements is indicative of their significance, and is the result of either evolutionary conservation in the case of homologs, or convergence in the case of analogs.\(^6\) As protein homologs evolve, greater conservation occurs at the level of structural motifs than sequence motifs, due to the direct correspondence between structure and function, and the resulting relationship between structure, fitness, and selective pressures. Sequence is conserved to a far lesser extent, and there are numerous examples of proteins which exhibit no detectable trace of sequence similarity, but which adopt similar structures, contain identical or related amino acid residues at their active sites, and have similar catalytic mechanisms. Observations of such shared features are consistent with the notion that these proteins are either homologous or are the product of functionally driven convergence of unrelated sequences. The latter explanation supposes that the observed structural similarities are strictly required for these proteins to perform the same or similar functions and evolved from entirely unrelated ancestors.\(^5\) (Note: convergence is meant here in the most general sense - the presence of common basic chemical or structural properties in the absence of common ancestry; in other words, an analogous relationship. For example, stabilizing disulfide bonds or utilization of the most common catalytic residues.) However, these explanations are mutually exclusive, and observations of common structural features known to be functionally relevant in certain molecules do not allow a determination to be made as to whether the true relationship is one of homology or analogy, if either; they are only consistent with the notion that one of these types of relationships may exist. It is possible that common structural features can be explained by genetic drift in the absence of functional conservation, or even by random chance, depending on their complexity.
Regardless of a protein’s evolutionary history, the identification of structural motifs with previously established function in a protein of unknown function may provide information about the protein’s biochemical role and potentially indicate its cellular and physiological significance. In the case that commonalities are a product of drift without functional conservation, their presence is not informative with regard to function. Even so, in the absence of evidence inconsistent with functional conservation, the presence of motifs known to be functional in other molecules suggest the possibility that their corresponding function may be present as well. Therefore, the detection of known functional motifs in structures of unknown function may serve as a hypothesis generating mechanism for inferring potential function. The basis of such functional inference is the extent to which a motif is conserved between proteins with a common function. The most well-known example of the relationship between structural motif similarity and common function is the Ser-His-Asp catalytic triads found in a variety of phylogenetically related and unrelated hydrolases. Numerous analytical methods have demonstrated the significance of this relationship by correctly inferring hydrolase function with high accuracy and sensitivity for both homologous and analogous proteins based on the detection of the catalytic triad motif. The predictive potential of this general approach has been validated for a variety of functions and motifs, and serves as the theoretical basis for a multitude of computational strategies for assigning function to structures predictively.

As of December, 2013, there were more than 95,000 structures in the Protein Data Bank (PDB), thousands of which are identified as having unknown function. These figures are indicative of the large gap which exists between the ability of structural genomics initiatives to resolve protein structures and the capacity to functionally annotate these structures. The emergence of this bottleneck has prevented the realization of the full potential of available
structural data to advance biological and medical research.\textsuperscript{14, 15} To address this issue, computational methods have been developed for assigning protein function on the basis of structural analyses. Various strategies have been employed, including evolutionary trace methods, methods maximizing superposition, graph-theory based methods, functional template matching, and machine learning techniques.\textsuperscript{2, 16-20} Specific approaches can be divided at a general level into two categories: those which search a novel structure against a reference set of predefined motif templates with known corresponding functions, and those which identify other attributes of the structure that are known to be associated with a function in other proteins (e.g. recurrence, residue propensity, conservation, etc...). The best representatives of the former category, such as JESS\textsuperscript{16}, are the most useful of any of these programs in terms of predictive accuracy and sensitivity, and their relative performance is a function of the quality of the motif templates and the algorithms used for detecting corresponding motifs in novel structures. Torrence \textit{et al.} provide a thorough assessment of these approaches.\textsuperscript{2}

ProMOL is a computational structural analysis program which uses a library of motif template definitions to search a given structure for features known to correspond to functions in previously annotated molecules.\textsuperscript{3} The components of ProMOL analysis are the target structure, the motif template library, and the detection algorithm. ProMOL is a plugin for the molecular visualization and analysis program PyMOL, and uses various functions available in PyMOL in conjunction with its own distinct capabilities to perform analyses. Users specify target structures with a PDB ID, and the corresponding PDB file is downloaded from the Protein Data Bank via an FTP server at the time of analysis. The PDB file is loaded into PyMOL and the resulting structural representation is the subject of subsequent steps in the analysis. The motif library consists of template definitions derived from various sources. A template is defined as the amino
acid composition and relative spatial positions of the key catalytic residues associated with the active site of a structure. Currently, motif templates are derived from three sources: JESS\textsuperscript{16} templates available from the Catalytic Site Atlas,\textsuperscript{17} a set of templates constructed by ProMOL developers and made available for download from the source code repository used to distribute ProMOL, and a set of user defined motifs. JESS templates are constructed programmatically by converting the specifications available from the Catalytic Site Atlas into the format used by ProMOL, and are included in the motif library that comes with ProMOL. These templates were originally based on a representative motif for a given protein family, and were defined by contributors to the Catalytic Site Atlas. The motifs defined by the ProMOL developers and included in the template library are constructed using a “Motif Maker” function of ProMOL, and are based on representative examples of motifs from various protein families. User defined motifs are also constructed using the ProMOL Motif Maker, and the specifications of these motifs are up to the user who creates them. The motif detection algorithm used by ProMOL currently utilizes five parameters: the relative spatial relationships of the residues in the template compared to the relative spatial relationships between potential matching residues in the target structure; the root mean square distance (RMSD) between alpha carbons of the template residues and the corresponding atoms in the potential matching set of residues in the target structure when the template is optimally aligned; the RMSD between the corresponding alpha and beta carbons of those same residues; the RMSD between all corresponding atoms in the residues; and the Levenshtein distance between the template motif and the set of potentially matching residues in the target structure.

The detection process used in ProMOL analysis proceeds sequentially for all motifs in the library, with three basic phases of comparison for each motif. The first phase is the
identification of potential matches in the target structure. In this phase, all residues in the target structure that are also present in the motif template are identified, and their spatial positions relative to one another are compared to the spatial positions of the residues in the template relative to one another. At the time of the analysis, the user specifies a Precision Factor, which is a threshold distance in Angstroms. Sets of residues with relative spatial positions matching those of the relative positions of the template residues to within the Precision Factor are selected as potential matches. Phase two of the analysis examines only these potential matches. The Levenshtein distance between the potential match and the template is calculated as the number of differences in component residue identities. Potential matches with Levenshtein distances below a threshold value are identified as positive matches. Finally, phase three of the analysis calculates the minimum RMSD between all atoms in the template and their counterparts in the match. This final phase has been made optional due to the significant computational demands and corresponding increase in runtime required to complete it, and it is possible to calculate these values for individual matches after the initial results are generated. Results are presented as a list of matches, their Levenshtein distances, and the RMSD of the alpha carbons only, the alpha and beta carbons, and all constituent atoms. Additionally, results may be visualized individually as the template motif optimally aligned with the matching structural feature of the query molecule and superimposed in three dimensions.

The central motivation for the development of programs which make functional assignments predictively based on structural analysis is to address the bottleneck resulting from structural genomics initiatives generating structural data faster than it can be annotated by traditional means. The rationale underlying this motivation is that computational functional assignment can be used to process large quantities of data quickly and efficiently compared to
traditional laboratory techniques. This processing generates hypotheses about the functions of unannotated structures which can be tested with conventional biochemical assays, thereby providing rapid, low cost, narrowly focused direction for ongoing, resource intensive, in vitro functional assignment efforts. With respect to this goal, ProMOL is deficient in that it requires an unnecessarily large commitment of time and computational resources, to such an extent that extremely large scale analysis - on the order of thousands of structures analyzed with template libraries representing hundreds of motifs - is not practical using personal desktop computers. This limitation can be attributed primarily to inefficiencies in the program design. Specifically, a major weakness in computational efficiency in ProMOL is the result of redundant analyses being common and difficult to avoid. ProMOL does not store results in a way that is accessible programmatically at any point after the results are initially provided to the user or by any instances of the program other than the one used to conduct the analysis at the time that it is done. Results are stored in text format as a comma separated value (CSV) file and visualized in ProMOL immediately after they are generated, and are inaccessible to the application thereafter. The consequences of these limitations manifest when a motif template library is added to or changed in any way; multiple users run analyses which include any of the same motif-structure pair comparisons; and when a user wishes to visualize or regenerate output from previous analyses. In each of these cases, detection computations for some number of template-target pairs must be performed multiple times.

Any given iteration of the detection algorithm for a target structure and a motif template requires substantial computation - on the order of seconds for medium to large size target structures on a typical desktop computer, if RMSD values are calculated. This is a concern that applies to many template-based structural detection programs by virtue of the necessary analysis.
Spatial comparisons in three dimensions require significant calculations, and runtime is dependent on the size of the target structure and the parameters used for template detection. While the case of one template being run against one average size structure completes quickly, the fact that useful template libraries contain hundreds or thousands of motifs and the PDB contains tens of thousands of structures - thousands of which lack functional annotation - means that analyzing a large portion of these structures would require thousands of hours. This issue is exacerbated by the fact that thousands of new structures are added to the PDB annually, and new motif templates are created and added to the ProMOL library regularly.

A second issue pertaining to ProMOL in its current form - and to other, comparable programs - is that a large amount of results data is not only difficult to generate, but difficult to work with and analyze computationally. ProMOL provides no means for programmatic analysis of results that have been exported to CSV files, and CSV files for the analysis of even a single target structure often consist of hundreds of cells containing values. This format is unworkable on a large scale without some means of automated computational processing that is designed with the ProMOL-specific result format in mind. A consequence of the current situation is that statistical analysis of results on a substantial scale is not possible. As a result, the significance of results cannot be easily quantified, and meta-analysis, which may yield useful information about motifs, protein families, or detection parameters, cannot be performed. The inability to quantify the statistical significance of results undermines their validity,¹ and the inaccessibility of metadata means the potential utility of large quantities of analysis data generated over time cannot be realized.

Finally, at least two desirable applications of the data generated by ProMOL are currently unavailable due to the format in which the data is stored. First, the question of the extent to
which a motif is conserved over a large number of structures cannot be addressed. Results generated by ProMOL for a family of proteins contain quantitative measures of the variability of a motif for each individual structure. Therefore the aggregate of these data indicates the variability of a motif across the entire family. This is significant because the extent to which functional motifs are conserved is the basis of the predictive power of template-based computational structural analysis applications. ProMOL results contain the information necessary to quantify motif variability, but their format does not permit such analysis without the development of additional software for processing ProMOL specific CSV file contents. Secondly, the results contain the information necessary to add a quantitative component to functional predictions, yet this is currently not a capability of the program. ProMOL and comparable structural analysis programs currently infer a type of enzyme function, but do not attempt to quantify the predicted activity. By comparing published measurements of the activities of various enzymes in a family to the variability in the functional motif corresponding to that activity, it would be possible to identify correlations between marginal structural variations with variations in activity, if they exist. This type of analysis could be the basis for a more detailed level of functional prediction and analysis, but requires analytical capabilities not presently implemented in ProMOL.

Fortunately, the design of ProMOL lends itself to the straightforward extension of its functionality to incorporate integrated access to a communal results database, which is conducive to a model of usage that can address all of the weaknesses of the program discussed thus far. The accessibility of ProMOL as free, open source software, the number of active developers, and the flexibility of its design make it possible to address the computational demands of large scale analysis and the inherent inefficiencies of the program by implementing a type of crowdsourcing
model based on a communal results database accessible by multiple instances of the client
program operated by a community of users. There are two core components to this strategy.
The first is that multiple users contribute to a database of results that is shared among the
community of users of which they are a part. This allows for the computational resource
requirements of analyzing many structures to be distributed over many independent instances of
the client program running on multiple computers, as opposed to a single user’s program running
on a single machine. The pooling of results from numerous sources makes the analysis of a
much larger number of results feasible by division of labor. The second component of this
strategy is to eliminate redundant computations. Instances of the client program will draw on
existing results in the communal database whenever possible, so that every individual template-
target pair will only be analyzed once. Marginal changes to the template library or the PDB will
require only that the resulting novel template-target combinations be analyzed. The combination
of these two improvements makes analysis of the entire PDB with a large template library
feasible with a relatively small number of contributing users. In addition, users will have much
faster access to results within the client application, and these can be readily visualized or further
analyzed in PyMOL.

The existence of a sizeable database of results accessible programmatically by the client
application, or directly by the user group operating the database, extends the functionality of
ProMOL to address the issues of statistical analysis of results and availability of metadata. Basic
database query functions can be used to easily quantify the statistical significance of results for
the entire program, individual motifs, and individual structures. Overall detection rates, false
positives, and false negatives can be calculated quickly and easily by comparing the contents of
the database to available lists of protein family members. Result attribute averages and variances
for motifs, structures, and sets of structures can also be determined, further characterizing the significance of results. Similar calculations could be applied to understand how the different match detection parameters correlate to the quality of results, and to determine if certain parameters are more informative under certain circumstances. These features would address a major issue with the validity of ProMOL results which exists currently, and would provide a rigorous method for prioritizing the expenditure of conventional laboratory resources on the confirmation of ProMOL functional predictions. Standard database queries also provide a basis for programmatic generation and analysis of other metadata that could be used to assess the extent to which motifs are conserved, indicating their potential predictive capacity. The same data could be used to determine the correlation between structural variations and variations in enzyme activity, allowing for a quantitative component of functional inferences to be calculated.

Materials & Methods

ProMOL is an open source structural visualization and analysis software application written in Python. It is currently available and under ongoing development. It is a plugin for the widely used molecular graphics system PyMOL. Binaries, source code, and documentation are available for both programs online, for Windows, Mac, and Linux operating systems.

This project consists of the design and implementation of a MySQL database for use with the ProMOL client application as well as modifications integrating database functionality into the existing source code. There are six main modules of code associated with this work, all of which are available as documented open source files online. Minor alterations to existing files in the core ProMOL package were also made, and are included, with documentation, in the latest open source release. The six major code components implemented for the database and its
integration into the client program are described here, and a model of usage is presented. All
database features are optional in the client program, so no connection to any database is required
for normal operation with non-database features. The database features must be turned on by the
user, and then are limited by the user’s access to the database they connect to, which they specify
when the database features are activated.

The database design itself is implemented as an SQL script, which generates a relational
database with the appropriate architecture. This was used by the ProMOL developers to
construct an instance of the database that all ProMOL users may draw results from automatically.
Initially, contributions and custom query access to this instance of the database is restricted to the
development team, in order to minimize vulnerabilities. Public access to query and contribution
features will be increased to whatever extent possible as integrity, stability, and security issues
are resolved. The development team works to maintain the database, updating it regularly to
include results for all PDB structures and all motifs in the default library. The contents of the
database are available separately with the rest of the ProMOL files for users who desire complete
access and their own copy of the data. Additionally, the script to construct the database may be
used by independent communities of users to create their own shared results database over which
they have complete control. As a default, the ProMOL client application draws results from the
database hosted by the developers when the database features are activated, and does not attempt
to contribute results generated by the client application unless credentials associated with known
developers are provided.

The database design consists of eight tables and stores information about users, active
sessions, structures, motif sets in the library, individual motifs, general results, specific result
details describing residue information, and specific result details describing RMSD information.

The architecture is represented in Figure 1.

Figure 1a - The relational architecture of the database. The six tables are given here with their associated attributes. Attributes contained within the same red rectangle depend on one another.

Figure 1b - The tables composing the administrative portion of the database. These tables are related to each other by account/user name, and are not related to any other tables in the database.

The credentials table (Figure 1b) contains usernames, passwords, and permission levels associated with each account. The passwords are encrypted in the database using the 128-bit
AES implementation native to MySQL. The administrators of the database enter these values manually. Credentials are used by the ProMOL daemon module to establish secure sessions. Credentials are passed by the client with every request. The first request - consisting only of credentials and a request to log in - is used to create a persistent unique identifier called a token with a call to the Session Manager module. Tokens are created using the uuid4() method of the uuid class in Python 2.7, which ensures that they are unique and unpredictable. The token is passed back to the client and is included in all subsequent requests. Until the user logs out - or the session expires - the token, username, and password, allow the client to have requests processed by a Session object with functionality defined by the user’s permission level. User names are primary keys in this table, meaning they must be unique and ensuring that a single user has a single permission level and password. The Sessions table stores tokens, the username associated with the token, and timestamps recording when the Session was created and when it will expire. The expiration timestamp is used by the Session Manager to delete Sessions after an appropriate period of time. The token is used to keep track of individual Session objects. Tokens are primary keys in this table, and must be unique, and user names are foreign keys and must exist in the Credentials table.

The Structures table contains entries for individual structures, and consists of a PDB ID, the dates of the most recent analyses with various subsets of the motif library, and the total number of distinct motifs detected in the structure. This information constitutes a portion of the descriptive metadata contained in the database. None of the values are used in queries or submissions by the client, and instead are intended to be used by directly accessing the database with MySQL. The primary key is the PDB ID of each structure, assuring that each structure only has a single entry in this table. The values are refreshed by the Database Manager every time
someone logs in and every time a submission is made to the database. The results table is referenced in order to derive the values contained in this table.

The Results table contains entries for unique results of individual motif detection analyses, and consists of a unique identifier, a structure ID, a motif ID, the precision factor used in the analysis, the algorithm version used in the analysis, the result of the analysis as either a Levenshtein distance or an indication that no match was found, and the date of the analysis. The unique identifier is arbitrary and used to connect results in this table with their specific details in the ResultSpecs and ResultRMSD tables. Entries are made by the client upon the completion of novel computations, as long as the user has sufficient permissions. The combination of parameters is the minimum set of details that defines a unique query. The unique identifier is the primary key, and the motif and structure ID are foreign keys referencing the motif and structures tables, and as such these values must be found in those tables at the time of submission. This is one of three tables in the database containing primary data, and is the main table referenced or updated during queries and submissions by the client.

The ResultsSpecs table contains entries for every residue from every result in the database that was a match, and consists of an arbitrary identifier for each data entry, the residue chain, name, and number, and the identifier of the corresponding result. These values are only meaningful when associated with entries in the Results table, and as such the result ID is a foreign key referencing the Results table. The contents of this table are the second portion of the primary data contained within the database, and provide the information needed to reconstitute and visualize positive results in the client.

The ResultRMSD table comprises the final portion of primary data, contains an entry for every result in the database that was a match, and consists of the RMSDs for the alpha carbon,
the alpha and beta carbons, all atoms, and the identifier of the corresponding result. Like the ResultSpecs table, the ID of the corresponding results are foreign keys referencing the Results table. These values are provided to the client in cases in which RMSD calculation is requested.

The Motifs table contains an entry for every motif in the current library, and consists of the motif ID, the name of the subset of motifs to which the motif belongs, the date the motif was added to the library, and the number of structures the motif has been identified in. The motifs present in this table are determined by the contents of two folders on the server: Motifs and UserMotifs. The folders must be updated manually, and the Motifs table is updated by the Database Manager whenever there is a submission or a log in. Results that are submitted must only pertain to motifs which are present in this table, and as a consequence novel motifs created by users and not submitted to the server by the database administrator cannot be a component of any result submitted to the database. Presently, this is enforced at the level of motif names, and unrecognized motif names will simply be passed over during the process of results submission. The subset of motifs to which an entry in this table belongs is a foreign key referencing the MotifSets table, and the motif IDs are unique primary keys.

The MotifSets table contains an entry for each subset of motifs in the motif library, and contains entries for the subset ID, the number of motifs in the subset, and the date that the subset was last modified. The subset IDs are unique primary keys. The values in this table are updated at the time of login and result submission by the Database Manager. This table constitutes the final portion of metadata contained in the database.

There is one module of code incorporated into the client program and four modules of code operating on the server that are responsible for the functionality of the database. The client side module is called the Admin Manager. It consists of a series of functions, which generate
and submit requests to the server and receive and process responses from the server. The Admin Manager is made available globally within the client program, and calls to its functions are made either directly via the graphical user interface (GUI) or indirectly as needed within the motif.py module, which is responsible for analytical computations and is the primary functional component of the client. The GUI has been modified to include a Database tab with four text entry boxes and two buttons. The text boxes provide input for the uniform resource identifier (URI), the database name, and the username and password. The URI and database name are populated by default with the URI of the ProMOL server hosted at RIT (https://oldmomtong.rit.edu/agrier/ProMOLHandler.py) and the ProMOL database on the MySQL server. The values in the text boxes are used when the user clicks the button to activate database access. The Admin Manager attempts to create a connection with the URI, and passes the database name and credentials so that the module receiving this request on the server can log into the MySQL database. The initial label and function of the button used to connect to the database changes once a connection is established, and in its altered form the same button logs out of the database and disconnects from the server. The second button associated with database functionality uploads legacy data in the form of CSV files to the database, and is still experimental and only available with administrator credentials. These two buttons are the only direct operations on the database that the user has access two. All other functions of the Admin Manager are used as needed by the analysis components of the client software. These functions consist of submitting queries, submitting entries, and triggering updates in the database. In each case, the relevant data being manipulated in the client as part of a normal analysis is converted into XML format according to the Simple Object Access Protocol (SOAP), and is submitted along with the user’s credentials and the current session token as a request to the server. All
requests, even one way submissions, are designed to elicit a response either containing the requested information or confirming successful receipt and processing of the request. All traffic between the client and server are processed through the Secure Socket Layer (SSL) cryptographic protocol to protect content. All data received by the Admin Manager is converted from XML format into Python objects according to SOAP, and all expected object types are enforced at the time of conversion to limit the potential of receiving unintended content.

The four modules of code on the server are designed hierarchically with simple procedural functions at the top calling increasingly complicated functions and classes below them. The highest level module is the ProMOLHandler. This is a single simple function which dispatches requests to the ProMOL daemon. SSL and client-server connections are handled directly by an Apache web server. The server is configured to run the ProMOLHandler for all requests directed at a specified port. Via mod_python, the ProMOLHandler provides a link between the Apache web server and the other Python modules associated with the ProMOL code.

All traffic passes directly through the ProMOLHandler and is dispatched to the ProMOL daemon. This module consists of a set of functions called by the handler according to a specific tag in an incoming request. It parses incoming requests and directly passes the relevant data to the SessionManager or ProMOLSesion modules as needed. It also passes the outgoing data through the web server to the client. Requests are processed via one of five functions: login, logout, query, submit, and update. Login passes credentials to the SessionManager which checks the credentials, creates a session, and returns a token and permission level. The token and permission level are passed back to the client to identify its session and define functionality. Logout passes a token to the SessionManager, which deletes the corresponding session. Query, submit, and update requests are processed by using a token and set of credentials to request a
Session object from the SessionManager, and then the data relevant to the desired operation are passed to the Session object with a call to the appropriate function. The daemon also persistently stores administrative credentials for the MySQL database, which it provides to the SessionManager and Session objects as needed, though it does not interact with the database directly at all.

The SessionManager is a class instantiated by the daemon and used to manage Session objects. It creates sessions when users log in, provides the specific session instance to the daemon when the specific client makes a request, and removes sessions when they expire or the user logs out. Upon instantiation by the daemon, the session manager connects to the database with administrative credentials, and it uses the Sessions and Credentials tables to keep track of sessions and check the credentials and permissions associated with requests. The session manager has three functions: requestSession, getSession, and retireSession. Requesting a session causes the manager to generate a session by submitting session information to the database. It returns the token and permissions associated with the session, and starts a timer to retire the session after a finite period (24 - 240 hours depending on account type). Getting a session confirms credentials and uses the identifying token to return a session object, which is used to execute functions. Sessions are retired automatically at the predetermined time by deleting their corresponding entry in the database. There is no theoretical limit on the number of concurrent sessions that are possible, however the rate at which the server can process requests may be limiting in practice if the number of concurrent users is very large.

Session objects are the primary functional components of the server-side program. They are provided to the daemon, which calls functions and passes the necessary information from the request to the session and from the session back to the client via the handler. Sessions consist of
three classes: the primary Session class which provides the three primary functions, and a resultsManager class and databaseManager class which provide the tools used by the three primary functions. The three primary functions are query, entry, and update. Query uses functions in the results manager to query the database for any results relevant to the current analysis being performed by the client. Entry uses the results manager to submit novel results generated by the client. Update uses the database manager to update metadata in the database when a session is initiated at login and whenever submissions are made. The results and database managers, as well as the session manager, employ the MySQLdb Python package to interact directly with the MySQL server. Additional details related to the implementation and purpose of these files may be found in the ProMOL source code and documentation. The series of panels in Figure 2 show how the various modules associated with the database functionality of ProMOL interact with one another for a set of common operations being performed by multiple users.

**Figure 2.1** shows a typical operating scenario in which a user (Client 3) has an ongoing session, another user (Client 1) is attempting to connect to the database, and a third user (Client 2) is not connected. Client 1’s admin manager program passes a request, which includes credentials, to connect and login to ProMOL Handler on the server, which is exposed by Apache. **2.2** shows that the Handler immediately dispatches the request and associated credentials to a new instance of the ProMOL Daemon, which generates a Session Manager object.
In 2.3, the Session Manager checks the credentials against those stored in the administrative portion of the database, specifically, the Credentials table. Upon confirmation of valid credentials, a session is created for Client 2 and input into the database, while at the same time a token and permissions information is passed back to the daemon. 2.4 shows the permissions and token data associated with Client 1’s session being passed back to the ProMOL Handler and ultimately through Apache.

In 2.5, the token and permissions data are passed, over the Internet by Apache via the handler, back to Client 1, which now has a session. 2.6 shows the beginning of how multiple simultaneous requests of different types are handled. Each client passes a request to the handler: Client 1 passes a query, Client 2 a login request, and Client 3 submits novel analysis results.
Multiple Daemons are spawned in 2.7 in response to the requests; the daemons store the data received with the request and create independent Session Manager objects. In 2.8, the Session Managers communicate with the administrative portion of the database. In the case of Client 2, a session is created. For Clients 1 and 3, the credentials and token are used to retrieve their existing sessions.

2.9 shows the subsequent independent processes associated with each request. A daemon returns Client 2’s token and permissions information to the handler, while the sessions associated with Clients 1 and 3 communicate directly with the core database to query and submit results, respectively. In 2.10, Client 2 receives its session information, while the daemons associated
with Clients 1 and 3 generate responses and pass them to the handler.

In 2.11, the Handler forwards the responses to Clients 1 and 3. 2.12 shows Client 1 passing a new request to the handler, which is a request to log out.

2.13 shows Client 1’s session information being passed from the handler to the daemon, which generates a session manager. In 2.14 the manager communicates with the administrative database to delete the session information of Client 1.
In 2.15, the Daemon generates a response and passes it to the Handler. It is forwarded to Client 1 in 2.16, which causes the admin manager in that client to change its state to reflect no longer being connected to the database.

2.17 shows the resting state of the modules after the above processes are completed.

While above figure details the flow of processes and data specifically between the various modules of code, Figure 3 provides a higher level view of the intended usage model of ProMOL, the database, and relevant third party resources.
Figure 3 - A representation of the intended model of usage for ProMOL with communal database functionality. Grey objects are user groups. Blue objects are data repositories controlled by independent third parties. Red objects are repositories controlled by the ProMOL development team. Green arrows show the flow of freely accessible data. Red arrows show the unidirectional flow of data or code files generated by the ProMOL development team. Orange arrows show the bidirectional flow of data generated by the ProMOL development team. The yellow arrow shows the bidirectional flow of data generated by independent third parties. All users who enable the database features will be able to retrieve applicable results from the public database(c) hosted by the ProMOL developers, which will contain continuously updated results for all structures in the PDB(b) and all motifs in ProMOL’s default library(e). Only members of the development team(a) will be able to contribute results to this database or perform queries other than those automatically performed by the adminManager module. As security concerns are addressed, public access to query and contribution features will increase. Independent user groups(f) will have the option of using a database of their own(g) which they can modify and control access to. The database construction code and ProMOL versions associated with the public instance of the database will be available to the public as free open source from ProMOL’s online repository(d).
Results

ProMOL has been updated to include all database functionality described here. There are two types of results associated with these changes: their effect on the performance and efficiency of the traditional operation of ProMOL, and the novel applications that they make possible. In terms of performance and efficiency, implementation of the database has decreased the runtime in the best and average cases, had no discernable effect of the worst case, and decreased the overall memory resources required by the ProMOL user community. Several novel applications are considered. The factors contributing to the runtime of ProMOL without utilization of database figures are present in Figures 4-7.

![Figure 4](image)

**Figure 4** - A plot of the runtime of a standard analysis using the ProMOL client without database features activated, as a function of the number of positive hits identified. The complete current motif library of 1193 entries, consisting of available JESS templates, accepted ProMOL templates, and promising user generated templates, was used and RMSD values were calculated. Representative structures of various sizes were selected, and a range of hits from 11 to 275 is observed. A weak linear trend is observed, indicating a positive correlation between the number of hits and the runtime of the program.
**Figure 5** - A plot of the runtime of a standard analysis using the ProMOL client without database features activated, as a function of the number of chains present in the structures queried. The complete current motif library of 1193 entries was used and RMSD values were calculated. Representative structures of various sizes with various numbers of hits were selected, and structures with between one and eight chains are represented. No significant relationship between number of chains and runtime is discernable.
Figure 6 - A plot of the runtime of a standard analysis using the ProMOL client without database features activated, as a function of the number of residues composing the query structures. The complete current motif library of 1193 entries was used and RMSD values were calculated. Representative structures with various numbers of hits and various numbers of chains were selected, and structures consisting of between 15 and 2044 residues were represented. A clear linear relationship between number of residues and runtime is observed, indicating a strong direct correlation between the size of a structure and the runtime requirements associated with analyzing it in ProMOL.
Figure 7 - A plot of the runtime of a standard analysis using the ProMOL client without database features activated, as a function of the number of residues composing the query structures, for two data sets utilizing different numbers of motifs. The complete current motif library of 1193 entries was used for one data set and a subset of 181 motifs was used for the other. RMSD values were calculated. Representative structures of assorted sizes with various numbers of hits and various numbers of chains were selected. Clear linear relationships are observed for both datasets between number of residues and runtime. Additionally, identical structures queried against a larger motif library had consistently greater runtimes, indicating that the number of motifs used in a query correlated directly with the runtime of that query.
Figure 8 - Runtime comparisons for assorted representative structures for a ProMOL client without database functionality enabled and for a ProMOL Client connected to and querying a database which has no results to contribute to the analysis. A partial motif library of 181 entries was used for each query and RMSD values were not requested. Runtime is observed to correlate primarily with the size of the structure being queried as measured by the number of residues composing it. There is no discernable difference in runtimes between the two client program scenarios.
Figure 9 - Runtime comparisons for assorted representative structures for a ProMOL client without database functionality enabled and for a ProMOL client utilizing a database, with various amounts of relevant data available from the database. RMSD values were requested in the query. Analyses were benchmarked under conditions in which: the database contained basic results - but no RMSD values - for 181 of the 1193 motifs queried; the database contained basic results for all 1193 motifs and no RMSD values; and the database contained the complete set of relevant results. In this figure only, runtimes are represented on a logarithmic scale. Analyses run without any relevant results present in the database were found to have the longest runtimes. When the database contained basic results for 181 motifs, the runtimes were 10-30% faster than when no applicable results were available. When basic results were available for all motifs, but no RMSD values were available, runtimes were typically 80-90% shorter. In the case that all requested results were available from the database, runtimes were typically 85-99% shorter.
Figure 10 - A plot of the runtime of a standard analysis using the ProMOL client with database features activated as a function of the number of positive hits composing the result. The complete current motif library of 1193 entries was used and RMSD values were requested. All relevant results were present in the database. Representative structures of various sizes with various numbers of chains were selected, and structures with between 11 and 275 hits were tested. A slight linear relationship is observed between the number of hits and the runtime, corresponding to a consistently weak positive correlation between these parameters.
**Figure 11** - A plot of the runtime of a standard analysis using the ProMOL client with database features activated as a function of the number of chains composing the query structures. The complete current motif library of 1193 entries was used and RMSD values were requested. All relevant results were present in the database. Representative structures of various sizes with various numbers of hits were selected, and structures with one to eight chains were tested. A slight linear relationship is observed between the number of chains and the runtime, corresponding to a consistently weak positive correlation between these parameters.
Figure 12 - A plot of the runtime of a standard analysis using the ProMOL client with database features activated as a function of the number of residues composing the query structures. The complete current motif library of 1193 entries was used and RMSD values were requested. All relevant results were present in the database. Representative structures with various numbers of hits and component chains were selected, and structures composed of 15 to 2044 residues were tested. A slight linear relationship is observed between the number of residues in the query structures and the runtime, corresponding to a consistently weak positive correlation between these parameters.
Figure 13 - Runtime comparisons for several sets of structures of various sizes, positive hits, and component chains, when the analysis of the structures in a set was run in a batch or individually. All three sets were analyzed with all results available from the database, and the set of the smallest structures was analyzed without the use of a database. The complete motif library of 1193 entries were used for each query and RMSD values were requested. Runtime is observed to correlate between batch queries and cumulative individual queries when the database is used for two of the three sets of structures. The third set of structures, which contains much larger structures, and five of them as opposed to three, exhibited significantly longer runtime when analyzed as a batch than the cumulative runtime of individual analyses. For the group of small structures analyzed as a batch without the database, the runtime was not significantly different from the cumulative runtime of individual analyses.
Figure 14 - The total memory requirements for user communities of between one and twelve individuals associated with representing the ProMOL analysis results of 100,000 typical PBD structures queried with the current complete library of 1193 motifs, including RMSD values, when the results are stored either on the communal database as described, or in CSV format locally for each user, as has been the practice. To store a single copy of these results in CSV format would require approximately 65 Gb of memory, estimated by multiplying the average size of these CSV files by the number of structures in the PDB. A single copy of the same results and all associated meta-data contained within the database would require approximately 200 Gb. The memory requirements when the data is stored in the database are constant regardless of the number of users. When stored locally, the memory requirements are proportionate to the number of users, with each requiring an independent copy of the results which occupies 65 Gb. For one or two users, local storage of the results require less memory. For three or more users, the database uses less memory.
<table>
<thead>
<tr>
<th>Motif</th>
<th>P_1n7n_4_2_2_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC Number</td>
<td>4.2.2.5</td>
</tr>
<tr>
<td>Homologs</td>
<td>9</td>
</tr>
<tr>
<td>True Positives</td>
<td>7</td>
</tr>
<tr>
<td>False Positives</td>
<td>1</td>
</tr>
<tr>
<td>True Negatives</td>
<td>90</td>
</tr>
<tr>
<td>False Negatives</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.78</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.99</td>
</tr>
</tbody>
</table>

**Table 1** - Using data accumulated in the database, the motif P_1n7n_4_2_2_5 was assessed for performance. Results of running it against 100 structures are shown here, with one of the structures being the basis for the motif itself, nine of the structures being known homologs, and 90 of the structures being known non-homologs. The motif returned hits for eight of the structures, and the numbers of true and false positives and negatives are shown, as well as the specificity and sensitivity for the motif based on these 100 analyses.
Table 2 - A detailed examination was made of the eight hits generated by the motif P_1n7n_4_2_2_5. This table shows the EC numbers, Levenshtein distances, and total RMSD values for each hit. The blue row identifies the structure on which the motif is based. The red row identifies the false positive, and the green rows identify the true positives.
Discussion

The ProMOL program is a plugin for the molecular visualization software package PyMol. ProMOL’s primary purpose is to predict the function of enzymes based on their structure, using computational analysis methods, in a way that requires less time and resources than traditional laboratory techniques. ProMOL results constitute testable hypotheses about enzyme function, which can then be verified via biochemical assays. ProMOL employs a template based motif detection algorithm to achieve this goal, by attempting to identify motifs of known function in enzymes of unknown function, thereby suggesting the function of the unknown enzyme. Computational functional determination of enzymes is an active area of research which is intended to address the bottleneck in functional genomics/proteomics between structural determination and functional annotation, and the corresponding accumulation of solved protein structures lacking functional assignment.

A substantial issue with ProMOL, as with all structure based function prediction programs, is that the required analysis is computationally intensive. The large and ever growing number of structures in the PDB makes analyzing all known protein structures virtually impossible on a desktop computer, given that a typical structure takes several minutes and that there are nearly 100,000 structures currently in the repository. This issue was addressed by the implementation of a shared, remote database, which facilitates a crowdsourcing strategy to analyze large numbers of structures and which eliminates the potential for unnecessarily redundant analyses. ProMOL originally yielded results in the form of CSV files stored locally on a user’s computer, which could not be reloaded in the application itself. This made large scale analysis of results difficult, and lead to redundant computations whenever 1) motif libraries were changed, 2) users wanted to visually analyze previously generated motif alignment results,
3) or a user wanted to generate results that a another, independent user had generated at some point prior. Implementation of the database provides a simple option for users of the client program to connect to a common database of results and retrieve all results previously generated by users connected to the database. This effectively eliminated redundant computations and in a large and ever growing number of cases dramatically increased the speed with which results could be loaded into the client.

The primary motivation for implementing the database features of ProMOL was to improve access to results. To this end, decreasing runtime was a priority. In order to understand primary factors which affected the runtime of the stand-alone ProMOL client program, a number of tests were conducted using a set of PDB structures with diverse properties including size, composition, and presence of known structural motifs. Test were also conducted with and without requesting RMSD calculations, and with differing motif library sizes. The results of these tests are shown in Figures 4 through 7. These results indicate that the primary factors affecting runtime are the number of residues present in the query structure and the size of the motif library used. There is a strong direct linear correlation between each of these parameters and runtime across molecules of differing sizes, compositions, and structures. This is as expected, as each motif must be analyzed sequentially against the structure, and each of those analyses involves checking all residues in the structure. Number of chains exhibits no relationship to runtime, and number of hits exhibits a weak positive correlation. Hits increase runtime because they require alignments between the motif and structure and subsequent RMSD calculations. Preliminary testing indicates that computing RMSDs in a single analysis as opposed to requesting only basic results increases runtimes by ~10% in a typical case, though
this is highly variable depending on the number of positive hits for which the values must be
computed and the number atoms in the motif templates associated with the hits.

To determine the effect of the database on the runtime of the client, tests were conducted
under circumstances of differing favorability. The first test, the results of which are shown in
Figure 8, was to compare the runtimes of the client when database functionality was not
activated against the runtime when it was connected to and utilizing a database which did not
contain any results relevant to the analysis being performed. For structures of all shapes and
sizes, there was no discernable difference in performance between these two conditions. This
not surprising, because the database features are not computationally intensive; SQL queries are
fast, and the only other potentially significant expenditure of time is in transferring data over the
internet between the client and server. The quantities of data are small enough and packaged
efficiently enough that no situation was found in which these delays could be detected against
background variations in runtime do to extraneous system processes.

When more favorable conditions were tested, in which some or all relevant results were
available in the database, runtime was found to decrease substantially. These findings are
detailed in Figure 9. Where results were available for some or all of the motifs queried, runtimes
were deceased by 10-90%, varying according to the proportion of motifs for which results were
present and to the size of the structure. When the RMSD was available in the database in
addition to the basic results for some or all of the motifs, runtimes decreased by >99% in some
cases. There appears to be a lower limit on the runtimes with the database of about 10 seconds,
and no analyses with or without the database were ever observed to take less than that amount of
time.
Figures 10 - 12 show how various properties of the query structure affect runtime when the database is used in a best-case scenario. A weak but significant positive correlation is observed between number of residues, number of chains, number of positive hits, and runtime. Residues, chains, and number of hits are not independent variables and are positively correlated with each other (a structure with more chains is likely to have more residues, making it more likely to hit on more motif templates by chance), so quantifying the significance of their relationships to runtime is difficult. Taken individually, any one of these variables accounts for approximately half of the variation in runtime that is observed between structures, in the average case, as indicated by the $R^2$ values in Figures 10 - 12. This is likely due to the fact that all of these features are associated with a greater quantity of results data. More residues almost always corresponds to more positive hits, and for each hit its location and RMSD values are identified in results data. More chains also increases the amount of data associated with positive results because the location of hits must be specified for more than one chain. These overall minor increases in the quantities of data associated with results mean that more data must be transferred between the client and server, and processed at either end. It should be noted that while the increases in runtime associated with these factors are real and observable in the test data, they may be considered essentially functionally negligible. For example, one query structure with 1,000 more residues than another is likely to take only about 10-20 seconds longer in the best case scenario with the database. This is as opposed to an expected difference in runtime of an hour or more between the same two structures when the database is not used. The overwhelming majority of the structures in the PDB have less than 1,200 residues, and the average structure consists of less than 700. This means that in almost all cases, if the database contains the analysis results, the runtime will be nearly constant, requiring less than a minute.
Figure 13 shows that submitting queries for analysis when results are available in the database as either batches or sequentially does not affect runtime in the average case, but that in cases where the batches are large, sequential query submissions are preferable, and this left up to the user. This is due to the packaging of the data transferred between the client and the server. On either end, the information comprising the query and the information comprising the results are processed into nested dictionaries and list structures in Python. These structures can accommodate all possible types and quantities of query and results data, and are converted to XML via SOAP for transmission over the internet and then back into Python objects again upon receipt in an algorithmically consistent and robust way, but the quantity and complexity of the XML data that must be generated, processed, and transmitted increases at a faster than linear rate compared to the size and complexity of the Python data themselves. The effect of this on runtime is noticeable with sufficiently large batch processes containing more than three or four large query structures and hundreds of results.

In Figure 14, the memory requirements associated with storing a very large quantity of ProMOL results data in a way that is accessible to multiple users using either the database or the previous stand alone client output are compared. This is meant to be a consideration of the entire ProMOL user community, and how much memory would be used if the entire PDB were analyzed and the results were available to everyone. These values are theoretical extrapolations of known values for smaller data sets. The memory used by a single shared copy of the results is less than the memory used by any significant number of copies of those results possessed by individual users. However, a single copy of this data on a single user’s computer is substantially smaller than the representation of the database containing the same data on the server. This is due primarily to the additional data - the metadata, relations, and administrative data - stored on
the database. In addition to the metadata described in this discussion that is specific to ProMOL results, MySQL automatically generates functional metadata to facilitate operations such as fast look ups and manipulations, features which CSV files do not have and which are vital to the performance of a database which must serve multiple users quickly. The “break even” point at which the total memory required for users to have access to complete analysis results is equal between a centralized database and independent local data storage is three to four users.

These data demonstrate the capacity of the database to make results more accessible to users by decreasing computational time and data storage requirements. The extent to which these benefits are realized depends on usage of the database and the accumulation of results. As ProMOL is used, and the database grows, the number of cases in which applicable results can be found will increase, which will make the average run faster. When computing novel results, the factors affecting runtime requirements are the number of residues in the structures being analyzed, the number of motifs being used in the query, and the number of hits which must be assessed and for which RMSD values must be computed. Runtimes when the database can be used to access results are nearly constant and are orders of magnitude faster than computing them from scratch, with some increases associated with the amount of data that must be retrieved, which is a function of the number of motifs being used in the query and the number of hits.

These developments make analysis of the entire PDB feasible over a matter of days or weeks by a small team of users with personal computers connected to the ProMOL database.

Another issue stemming from the storage of results as local CSV files is that computational analysis of large numbers of results was difficult: custom scripts would have to be written to answer fairly simple questions regarding things like motif accuracy or other metadata. This issue hinders the user’s ability to quantify and ultimately improve the accuracy of ProMOL,
and limits the utility and value of the large quantities of data that can be generated with ProMOL. Implementation of the results database resolves this issue by permitting straightforward and efficient programmatic access to all results in the database via MySQL. This allows accuracy statistics to be generated simply, in addition to permitting a wide variety of novel analyses to be performed on the ProMOL results dataset. Tables 1 and 2 provide an example of the utilization of these capabilities to both quantify the performance of a motif, and to suggest straightforward criteria for evaluating the quality of a result. Table 1 simply quantifies the performance of a single motif run against 100 structures of known function in terms of sensitivity, specificity, and true and false positives and negatives. Table 2 examines the Levenshtein distance and RMSD values for true and false positives for a single motif run against 100 structures of known function. From these data, it is clear that while the false positive was indeed erroneously identified as a hit, its Levenshtein distance and RMSD values were much different than any of the true positives. This suggests that positive results could be scored based on Levenshtein distance and RMSD values in order to identify results that are likely to be erroneous.

By being able to analyze the entire PDB and store the results in the database, the nature of ProMOL is changed dramatically. The client program becomes more of a lookup service than a computational engine. The accuracy of each motif template can be determined with a statistically significant sample size, and the characteristics of what makes a motif accurate or not can be determined using simple MySQL scripts, leading to better motif templates being implemented. These capabilities will allow ProMOL results to be ranked, so that testing the corresponding functional assignments can be prioritized with confidence. This further advances ProMOL’s ultimate goal, which is to make function assignment less resource intensive.
Given the preliminary results presented here and the vast array of potential new avenues of exploration using ProMOL, there are several promising directions for future work based on the database. Perhaps the most pressing is the analysis of the entire PDB. Once complete, the resulting dataset is expected to provide numerous novel avenues of investigation, in addition to serving as the basis for improving the performance of ProMOL. Subsequent computational analyses will only be necessary when marginal changes are made to the PDB and to the ProMOL motif library, making upkeep of the comprehensive set of results fast and easy. Once that is done, a results ranking system may be implemented, and better motifs may be developed.

Along similar lines as the implementation of a basic result ranking system based on accuracy statistics for a given motif, as well as statistics quantifying the error rates of hits as a function of Levenshtein distance and RMSD values, a more fundamental methodological concern may be addressed. ProMOL fails to incorporate evolutionary information into its calculations. The component of these analyses that drives up the rate of false positives is the frequency with which structural commonalities occur by chance or due to drift in the absence of functional conservation. The ability to discriminate random, false hits from true positives is fundamentally limited by the resolution of the molecular structures, the actual physical flexibility of active sites, and the tolerance in terms of spatial alignment that the ProMOL algorithm uses when identifying matches. The tolerance of ProMOL alignments can be controlled with the Precision Factor parameter, and the resolution of molecular structures is a product of technologies and experimental conditions beyond the scope of this project. The only ways for ProMOL to address this issue of random chance matches is to utilize sufficient precision in matching, quantify the false discovery rate of each motif, and to use motifs with sufficiently large numbers of residues (increasing their complexity and decreasing the probability of them
occurring randomly). These factors may be addressed without major modifications to the core algorithm in the form of a result scoring system. The other driver of false positives, drift, would require substantial changes or additions to be made to the core ProMOL algorithm. Drift may be addressed to some extent by incorporating phylogenetic information into ProMOL analyses.

Given sufficient information about the structures of evolutionarily related proteins, it may be possible to control for features that are the product of drift and are not informative with regard to function. Sequence data may contribute to this effort. A potential long-term goal may be to modify ProMOL based on these ideas in order to reduce the rate of false positives.

Finally, a concern related to the implementation of the database may be addressed in the future. ProMOL is freely available and open source, and the user community is expected to represent a broad spectrum of technical knowledge and ability. The database features of ProMOL require the installation of two fairly sophisticated Python packages with the client program: PyXML and the Zolera Soap Interface (ZSI). These programs require compilation of files written in C, and their installation is somewhat dependent on the specific configuration of a user’s system. This can lead to some difficulty in the installation process, which is markedly more involved with the database features than without. It is desirable that these packages or some of their capabilities be better integrated in the ProMOL package, perhaps with an automated installer, so that usage of the database is simple for users without significant technical background.
References


