Efficient Algorithms and Architectures for Protein 3D Structure Comparison

Anuj Sharma*

National and Kapodistrian University of Athens
Department of Informatics and Telecommunications
asharma@di.uoa.gr

Abstract. In this dissertation, the problem of comparing proteins based on their three dimensional structure is studied. With new Protein Structure Comparison (PSC) methods continuously emerging and no clear method of choice, *Multi-Criteria Protein Structure Comparison* (MCPSC) is commonly employed to combine methods and generate consensus structural similarity scores. We developed methods to allow users to perform MCPSC efficiently, by exploiting the parallelism afforded by todays desktop computers. We implemented these methods in a Python based utility, *pyMCPSC*. We demonstrate how *pyMCPSC* facilitates the analysis of similarities in protein domain datasets. Finally, we exemplify the power of *pyMCPSC* on several datasets and show how it can help users gain insights about their datasets.

1 Introduction

For the last three decades the comparison and alignment of protein structures has been used extensively in computational biology, because naturally occurring protein fold in three dimensional space and the resulting structure has a strong correlation to its function [1]. Conservation of proteins is known to be much higher at the structure than at the sequence level, therefore structural similarity is essential in assigning functional annotations to proteins [2]. Function assignment is typically achieved by developing a template of the functional residues of the proteins and then aligning the template with complete known structures [3]. Structural comparison approaches are also increasingly employed in drug repositioning [4]. Protein Structure Comparison (PSC) methods are used to identify proteins with similar binding sites all of which then become potential targets for the same ligand [5]. All these important applications require the structure of one or more proteins (queries) to be compared against a large number

* Dissertation Advisor: Elias S. Manolakos, Professor

of known protein structures (one-to-all or many-to-many type comparison) to identify protein pairs with high structural similarity.

Computational demands in PSC are a result of three pertinent features. Firstly, pairwise PSC has high computational complexity due to the NP-Hard nature of the problem. Secondly, classification of newly discovered protein structures, for the purposes of ascertaining functional properties, requires comparison with large and fast expanding databases [6]. Thirdly, the lack of consensus on a single method has led to a trend in the domain to provide results from more than one structure comparison method [7]. The advantage of such an approach, Multi-criteria Protein Structure Comparison (MCPSC), is that it does not call for determining the superiority of an approach over another, but integration of several protein structure comparison methods into a unified tool. The approach banks on the idea that an ensemble of classifiers is likely to yield better performance than any of the constituent classifiers [8].

In published literature instances of the use of distributed computing platforms, such as clusters of workstations (COWs), computer grids and cloud, can be found for meeting this computational demand [9]. One such implementation, available for use to the community, the ProCKSI server [10], is an online resource for performing all-to-all MCPSC experiments. Results of the experiments returned to the user include individual PSC method scores as well as a consensus MCPSC score. While the ProCKSI server provides an excellent one stop resource for designing and running all-to-all MCPSC experiments, it is limited in the size of the data (upto 250 protein domains) and is not extendable with PSC methods of users choice. In general, distributed solutions, specifically Grids, suffer from problems such as extensibility, maintainability and fault tolerance.

These parallel processing architectures have become more readily available and instances of their use are beginning to appear in the broader field of biocomputing. These architectures can in principle be used additively to meet the ever increasing computational demands of MCPSC by complementing already in use distributed computing approaches [11]. It is therefore critical that effort be expended to utilize this desktop scale parallelism. This will allow problems of a scale that could only be tackled by distributed platforms, to be carried out on commodity hardware and perhaps even more efficiently [12].

As established, protein structure comparison is a well developed field of research with a large body of published literature and active current interest with new techniques being developed continuously. We point out some of what we believe to be open challenges for large-scale MCPSC:

- *Efficient use of modern parallel architectures*: Existing works focus on improving pairwise PSC or on application of distributed resources to the many-to-many PSC scenario. However, the parallelism afforded by modern parallel architectures has not been extensively explored or efficiently utilized especially for many-to-many PSC.

- *Biologically Relevant Clustering and Classification*: Defining and identifying classification of different protein structures is still an unresolved

problem. Structured analysis of gain from Multi-criteria PSC towards resolving this problem is needed.

- *Leveraging Structure Comparison*: Generating large-scale MCPSC results currently requires large distributed resources. Since a higher amount of computational power is available locally to researchers, software needs to be made available so that they can leverage such results in their every day work. Thus, effort is needed to deliver high quality easy to use software that can utilize desktop parallelism.

In this work, three PSC methods – TMalign [13], CE [14] and USM [15] – were ported on the Intel Single-chip Cloud Computer (SCC) [16], a Network on Chip (NoC) based many-core processor with 48 Pentium cores organized in a mesh network. In order to facilitate porting PSC methods to the Intel SCC an algorithmic skeleton library called *rckskel* was developed. The library was designed to provide basic functions needed for exploring hybrid parallelization strategies, for speeding up protein structure comparison algorithms. The result was an application that uses the three PSC methods to perform all-to-all protein structure comparison. The application makes use of all SCC cores (available at run-time) to run multiple pairwise PSCs concurrently.

The MCPSC implementation for the SCC was used to conduct several experiments to study its speedup characteristics. Each pairwise PSC with a single method was considered as a job, which is the most fine-grained work distribution setup that can be achieved, resulting in $N \times N \times M$ jobs, where N is the number of proteins and M is the number PSC methods. In the experimental setup the lengths of the pair of proteins were used as a factor indicating the expected time complexity of performing pairwise PSC. Preliminary results, with TMalign alone, showed that using the 48 cores of the SCC in a distributed setting (master running on a separate processor) is not efficient, giving a speedup lower than a factor of 3x. Several experiments conducted with load balancing strategies revealed that dynamic round-robin job distribution outperforms static job partitioning schemes. It was observed that a near linear speedup can be achieved as the number of slave-cores is increasing, which suggests even higher speedups are possible with bigger NoCs [16].

Prototype FPGA implementations of two PSC methods – TMalign and USM – were also developed. A feasibility study was conducted to ascertain if such implementations are viable. Using FPGAs for speeding up PSC was not found to be feasible because: a) either the parallelizable parts of PSC methods are a very small factor of the overall method as in the case of TMalign, or b) the software implementations are already highly optimized and no significant gains can be achieved from the hardware implementations, as in the case of USM. A byproduct of our initial experiments into the development of an efficient superposition subroutine for the FPGA, was a Python module for structure superposition, QCPSuperimposer, which was submitted and accepted as a module in BioPython v1.66 [17].

A multi-threaded implementation of the MCPSC software was developed for comparison with the many-core implementation. Experiments were designed

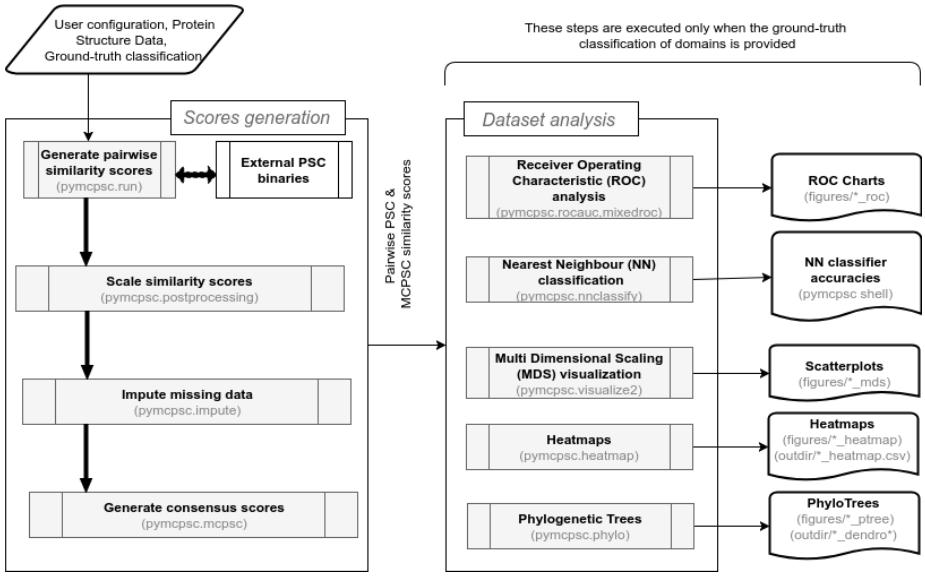
to assess how the MCPSC problem scales with increasing number of cores on a modern multi-core CPU and to compare this with the speedup observed on a many-core CPU. The multi-threaded implementation makes use of OpenMP for introducing threads and uses shared memory constructs to replace the message-passing communication handled by *rckskel* in the many-core implementation. Our experiments showed that, when running on a quad-core Intel i7 (with 8 hyper-threaded cores), speedup is observed up to the 4 threads configurations thereafter a steep speedup drop (efficiency loss) is observed. This was attributed to the fact that the work-load placement is not suited to taking advantage of the super-scalar structure of the CPU which requires varied workloads to deliver higher performance. Comparison of the many-core and multi-core CPU throughputs, on several datasets of varied sizes, showed that while the i7 is superior in raw times it only achieves a 4x throughput (in terms of pairwise protein structure comparisons per second) as compared to SCC while it runs at 7.5x the frequency [18].

Finally, experiments with a very large protein domain dataset (3,213,631 protein domain pairs) using a utility we developed, named *pyMCPSC* , containing five PSC methods - TMalign, CE, USM, Fast [19], GRalign [20] - were carried out to determine the characteristics of consensus-based MCPSC. The experiments were designed to run only on a multi-core CPU due to the memory limitations of the Intel SCC many-core processor. As a software architecture, *pyMCPSC* is organized into several modules called in sequence by the main entry point. An overview of the processing sequence is shown in Figure 1. Qualitative analysis using Receiver Operating Characteristics (RoC) curves revealed that simple consensus schemes, such as using the average score, resulted in MCPSC performing near-optimally in comparison to its component PSC methods. This was also validated in terms of its structural classification ability using a Nearest Neighbor approach. A key observation of these experiments was that MCPSC closely follows the best performing PSC method, which is of importance in the absence of the ground-truth in a domain where no single method is considered to be complete or superior. Finally, we visualize the dataset in domain and topology spaces and show that such visualizations reveal interesting information about presence of correlations and clusters within a large dataset.

2 Results and Discussion

We will demonstrate the use of *pyMCPSC* using protein pairs obtained from the Proteus dataset. PSC scores were obtained for these pairs and analyzed as discussed in the previous section. The number of pairwise PSC jobs processed per PSC method is actually one half of this value because of the symmetry of the PSC scores matrix, however the post processing and performance calculations are performed with the full matrix. The PDB files, the ground-truth SCOP classification and the pairwise domain list as well as the experimental setup are included in the *test* folder of the downloadable sources. *pyMCPSC* generates performance results for three sets of domain pairs, defined as follows: a) Original

Fig 1. Schematic overview of the architecture of *pyMCPSC*.



pyMCPSC is organized into several modules, each one implementing a specific functionality. The main entry point of the utility drives the sequence of activities shown. Similarity scores are generated for all protein pairs using the executable binaries of the included PSC methods. Subsequently the scores are scaled, missing data (similarity scores) are imputed (local average fill) and consensus MCPSC scores are calculated for all domain pairs. If the user has supplied ground-truth domain classification information, then comparative analysis results are also generated based on the similarity scores. The modules where the respective functionalities are implemented are specified in parenthesis.

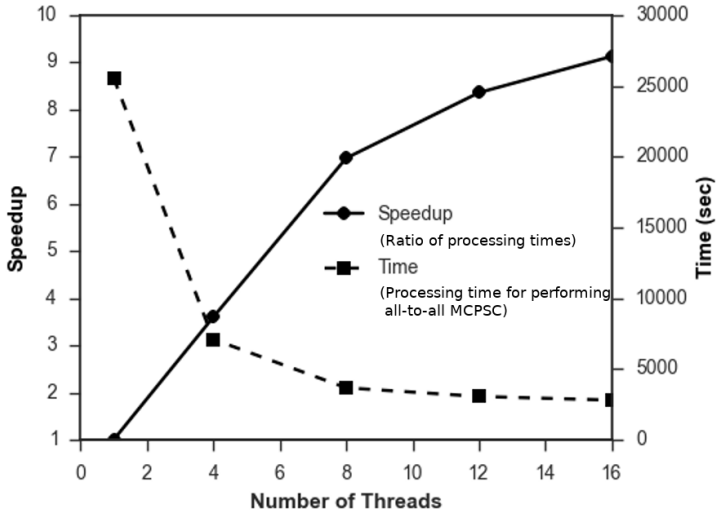
Dataset: all pairwise scores but with missing values, b) Common Subset: excludes pairs where any pairwise PSC score is missing and c) Imputed Dataset: missing values supplied by imputing.

2.1 Performing MCPSC on a multi-core processor

Using *pyMCPSC* we generated pairwise similarity scores (all-to-all) based on the 5 PSC methods and the 5 MCPSC schemes (M1 - M5) included in the utility by default, as well as the pairwise median MCPSC scheme. Experiments were carried out using multi-threaded processing on an Intel Core i7- 5960X "Haswel" 8-Core (16 Threads) CPU running at 3.0 GHz with 32 GB of RAM and an SSD running Linux. The Core i7 CPU features highly optimized out-of-order execution and HT (Hyper Threading), Intel's flavor of Simultaneous Multi-Threading (SMT).

The number of domain pairs for which scores were successfully generated varies among the PSC methods, with GRALIGN and FAST having the lowest

Fig 2. Speedup factor and total processing time for performing all-to-all MCPSC with increasing number of threads on a Intel Core i7 multicore CPU using the Proteus 300 dataset.



coverage. This is attributed to differences between the build and runtime environments, the thresholds built into the PSC method programs and errors in the structure files downloaded from the PDB. A speedup factor of 9.13 is achieved for end-to-end processing of the Proteus 300 dataset using *pyMCPSC* when $p = 16$ threads are used [21].

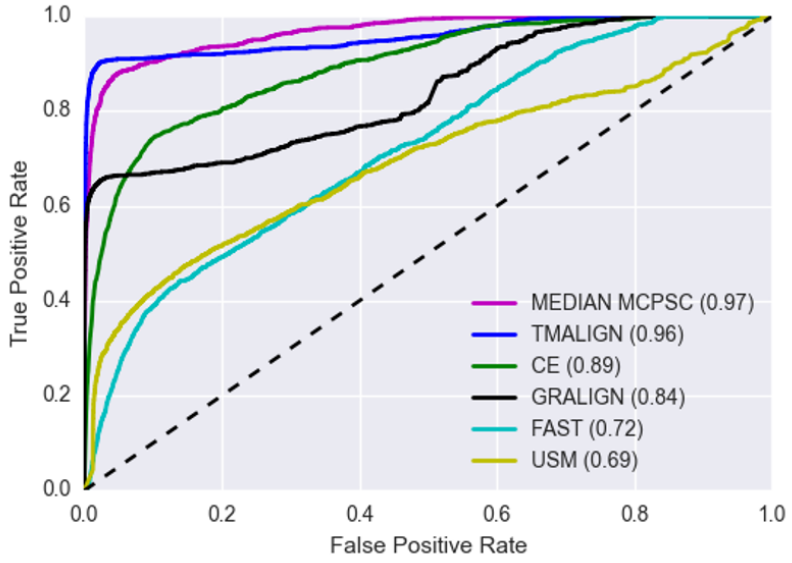
In Figure 2 we show the speedup factor achieved and the total processing time as the number of threads increases from 1 to 16. Nearly linear speedup is observed till the number of threads reaches the number of available cores of the CPU (8). The speedup continues to grow with the number of cores even beyond that point, albeit at a slower rate. This analysis suggests that the emerging many-core processors with more than 16 cores could also be exploited by *pyMCPSC* to analyze very large datasets.

2.2 *pyMCPSC* generates quality consensus scores

Receiver Operating Characteristics (ROC) analysis can be used to compare the classification performance of MCPSC with that of the component PSC methods. *pyMCPSC* uses ROCs and corresponding Area Under the Curve (AUC) values for performance benchmarking if ground truth data is available.

In Figure 3, we see that for this dataset TM-align achieves the highest AUC among the five supported PSC methods. Moreover using the median MCPSC score matches or exceeds the AUC performance of the best component method. This actually remains the case even if we remove TM-align from the pool of the PSC methods and repeat the same analysis with the four remaining methods.

Fig 3. ROC curves of all PSC methods and the median MCPSC method using the Imputed dataset of pairwise similarity scores. The ROCs are generated at the SCOP Superfamily level (Level 3).



In reality, we do not expect to know which PSC method will perform the best for any given dataset. So as the results suggest, combining PSC methods to obtain MCPSC scores and then using their median as the final consensus score to assess similarities is an effective strategy.

2.3 *pyMCPSC* consensus scores can be used to accurately classify query domains

Nearest-neighbor (NN) auto-classification can be used to assess how well PSC methods can classify a query domain, given pairwise PSC scores and the structural classification of other domains. When a new protein structure is determined, it is typically compared with the structures of proteins with known SCOP classifications. Therefore, the accuracy of the PSC and MCPSC based NN-classifiers effectively reflects their ability to be used for automatic protein domain classification.

Distance matrices based on the PSC and MCPSC scores are used by *pyMCPSC* to perform NN domain classification. MCPSC based NN-classification matches or exceeds the performance of the best PSC method at all SCOP hierarchy levels, with and without data imputation Table 1. Moreover, whereas the classification performance of the five supported PSC component methods varies considerably for the same SCOP level, the performance of the five different MCPSC methods is consistent. This suggests that using *pyMCPSC* to implement

	Original dataset				Common subset				Imputed dataset			
SCOP Level	1	2	3	4	1	2	3	4	1	2	3	4
TM-align	1.00	1.00	0.99	0.99	0.74	0.57	0.57	0.57	1.00	1.00	0.99	0.99
CE	0.78	0.61	0.61	0.60	0.63	0.47	0.47	0.47	0.76	0.60	0.60	0.58
GRALIGN	1.00	1.00	1.00	1.00	0.74	0.57	0.57	0.57	0.89	0.89	0.89	0.88
FAST	0.20	0.08	0.08	0.08	0.19	0.07	0.07	0.07	0.20	0.08	0.08	0.08
USM	0.84	0.72	0.67	0.65	0.65	0.51	0.49	0.49	0.84	0.72	0.67	0.65
M1	0.99	0.98	0.98	0.98	0.73	0.57	0.56	0.56	0.99	0.99	0.98	0.98
M2	0.99	0.98	0.98	0.98	0.75	0.57	0.56	0.56	0.99	0.98	0.97	0.97
M3	1.00	1.00	1.00	1.00	0.74	0.57	0.57	0.57	1.00	1.00	1.00	1.00
M4	0.99	0.99	0.99	0.99	0.72	0.57	0.57	0.57	0.99	0.99	0.99	0.99
M5	1.00	1.00	1.00	1.00	0.74	0.57	0.57	0.57	1.00	1.00	1.00	1.00
Median **MCPSC**	0.99	0.99	0.99	0.99	0.74	0.57	0.57	0.57	0.99	0.99	0.99	0.99

Table 1. Fraction of domains correctly classified at different SCOP hierarchy levels using a Nearest-Neighbor classifier built with similarity scores produced by different PSC and MCPSC methods. In the SCOP hierarchy: Level 1 = Class, Level 2 = Fold, Level 3 = Superfamily and Level 4 = Family.

different MCPSC methods and then using their median score in conjunction with NN classification can provide trustworthy query domain auto-classification results. These results also highlight that in the absence of ground truth information and/or lack of prior knowledge as to the best PSC method for a dataset, MCPSC can be employed to accurately auto-classify new domains.

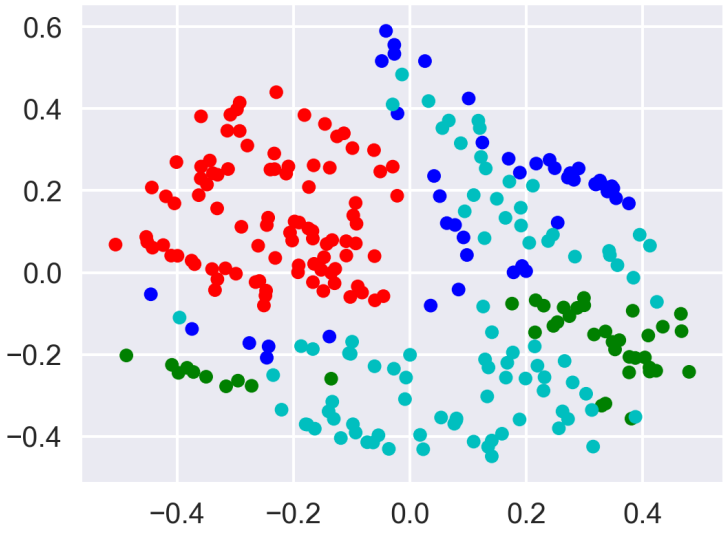
The results show that the best MCPSC method matches the performance of the best component method and the Median MCPCS based classification is almost always optimal, which makes median MCPSC a good choice for classifying query domains when prior knowledge about the best PSC method is not available. Moreover, the performance differences of the MCPSC methods are minor, suggesting that they are all quite robust to significant variations on the performance of their component PSC methods. The lower performance observed for all methods on the Common subset is probably a result of the small percentage of domain pairs for which similarity scores are available by all methods (less than 50%).

2.4 *pyMCPSC* reveals structural relations between domains

pyMCPSC uses PSC/MCPSC based distance matrices in conjunction with Multi-Dimensional Scaling (MDS) to generate insightful scatterplots of protein domain organization in the structural space. An $N \times N$, distance matrix D is constructed, with N being the number of unique domains in the dataset. Matrix element D_{ij} corresponds to 1 - S_{ij}, the pairwise scaled dissimilarity score of domains d_i and d_j, where $i, j \leq N$, are drawn from the imputed data set. Missing values $(N^2 - P)$ are set to 1. The value of 1 (max dissimilarity) is selected so that all domains appearing close in the visualization are in fact close to each other based on the selected method's score.

pyMCPSC uses matrix D as the basis for MDS to produce scatterplots of domains. This effectively assigns a 2-Dimensional coordinate to each protein domain constrained by the pairwise domain distances specified in matrix D. The resulting scatterplots can be used to visually explore a domains dataset, revealing

Fig 4. MDS scatter plot based on median MCPSC scores. Domains are colored according to their SCOP class (Level 1).



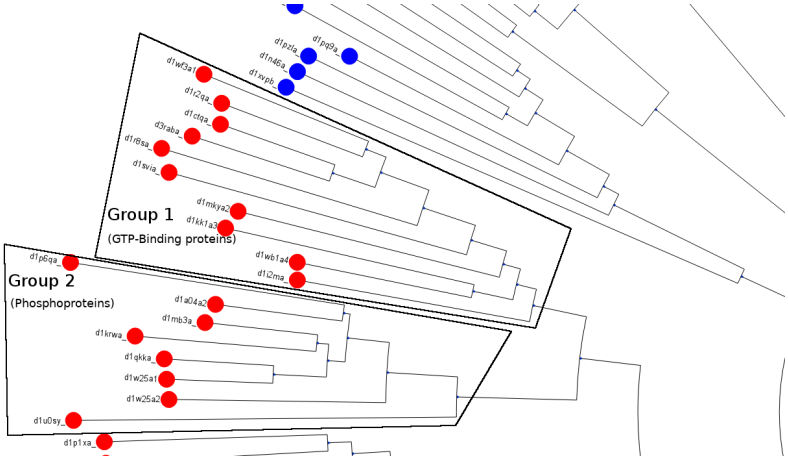
existing correlations. Figure 4 shows the layout of the domains of the imputed dataset in 2-D space resulting from MDS using the median MCPSC scores. Such a visualization produced by *pyMCPSC* suggests that for the given dataset the SCOP Class C domains (red color) exhibit higher interdomain similarity. This is in stark contrast to the domains of SCOP Class D (cyan color) which are diffused across the scatterplot. This observation is further evidenced by the Heatmaps also generated by *pyMCPSC* which in addition reveal finer level structure inside each class.

2.5 *pyMCPSC* can reveal functional relations between protein domains

pyMCPSC uses similarity score based distance matrices (D) in Phylogenetic trees to provide functional grouping of domains. *pyMCPSC* uses a Neighbor-joining algorithm from *dendropy* to create dendrograms and uses them to generate unrooted circular layout Phylogenetic trees. The goal is to create trees where the domains are separated into clades based on their function.

In Figure 5 we have marked two groups of domains belonging to different clades in the tree. The most common keyword for Group 1 is 'GTP-Binding' while for Group 2 it is 'Phosphoprotein'. The clades of the Phylogenetic Tree generated by *pyMCPSC* could therefore be used by a researcher to identify groups of domains (within the same SCOP class as in this example) that are

Fig 5. The unrooted Phylogenetic Tree based on median MCPSC consensus scores. Domains are colored according to their SCOP class (Level 1). Domains of the two clades that are marked belong to Class C but represent different functional groups.



functionally different.

3 Conclusion

As the number of protein structures grows we are faced with the important but complex task of assigning function to proteins as well as classifying them into biologically pertinent groups. The faster these tasks can be performed, the faster biologists and medical researchers can determine possible applications for the protein. The main focus of this thesis was on developing computational methods which facilitate fast and efficient Multi-criteria Protein Structure Comparison (MCPSC).

Results obtained from large-scale MCPSC, show that MCPSC delivers near optimal performance in terms of classification and clustering of proteins. This is largely because MCPSC scores tend to trace the best performing component PSC method. We believe that this is an interesting result because in the absence of the ground-truth data it is not evident which PSC methods is performing the best. Consensus based MCPSC scores can therefore be used to analyze existence of clusters within datasets of biological importance as well as to auto-classify protein (domains) into categories similar to SCOP/CATH.

Finally, more effort is needed to study further the complete structure space by carrying out even larger-scale experiments (with all the PDB for instance). Understanding the nature of the complete structure space can be catalytic in defining and identifying unique structural units that are recurrent between protein structures. Such a finding could help design better structure alignment metrics, as well as improve the ability of existing methods to categorize proteins

in a biologically relevant manner. Currently, domains are considered evolutionary units because they show similar activity even when extracted from a protein chain all together. Thus proteins with similar domains or with similar arrangements of domains are considered similar. However, the sequence diversity with the domains as recurring structural units is still huge and certainly worth exploring both for causes and consequences.

References

1. Laskowski RA, Watson JD, Thornton JM. ProFunc: a server for predicting protein function from 3D structure. Nucleic Acids Research. 2005;33:89–93.

2. Mills CL, Beuning PJ, Ondrechen MJ. Biochemical functional predictions for protein structures of unknown or uncertain function. Computational and Structural Biotechnology Journal. 2015;13:182 – 191. doi:https://doi.org/10.1016/j.csbj.2015.02.003.

3. Redfern OC, Dessailly BH, Dallman TJ, Sillitoe I, Orengo CA. FLORA: A Novel Method to Predict Protein Function from Structure in Diverse Superfamilies. PLOS Computational Biology. 2009;5(8):1–12. doi:10.1371/journal.pcbi.1000485.

4. Haupt VJ, Daminelli S, Schroeder M. Drug Promiscuity in PDB: Protein Binding Site Similarity Is Key. PLOS ONE. 2013;8(6):1–15. doi:10.1371/journal.pone.0065894.

5. Duran-Frigola M, Siragusa L, Ruppin E, Barril X, Cruciani G, Aloy P. Detecting similar binding pockets to enable systems polypharmacology. PLOS Computational Biology. 2017;13(6):1–18. doi:10.1371/journal.pcbi.1005522.

6. Chi P. Efficient protein tertiary structure retrievals and classifications using content based comparison algorithms. Columbia, MO, USA; 2007.

7. Barthel D, Hirst JD, Blazewicz J, Burke EK, Krasnogor N. ProCKSI: a decision support system for Protein (Structure) Comparison, Knowledge, Similarity and Information. BMC Bioinformatics. 2007;8:416. doi:10.1186/1471-2105-8-416.

8. Kuncheva LI, Whitaker CJ. Measures of diversity in classifier ensembles. Machine Learning. 2000;51(2).

9. Mrozek D, Malysiak-Mrozek B, Klapcinski A. Cloud4Psi: cloud computing for 3D protein structure similarity searching. Bioinformatics. 2014;30(19):2822–2825. doi:10.1093/bioinformatics/btu389.

10. Barthel D, Hirst JD, Blacewicz J, Krasnogor N. ProCKSI: a Metaserver for Protein Comparison Using Kolmogorov and Other Similarity Measures. BMC Bioinformatics. 2007;8:416.

11. Shah AA, Barthel D, Krasnogor N. Grid and Distributed Public Computing Schemes for Structural Proteomics : A Short Overview. Frontiers of High Performance Computing and Networking ISPA 2007 Workshops. 2007;4743:424–434.

12. Colmant M, Rouvoy R, Seinturier L. Improving the Energy Efficiency of Software Systems for Multi-core Architectures. In: Proceedings of the 11th Middleware Doctoral Symposium. MDS '14. New York, NY, USA: ACM; 2014. p. 1:1–1:4. Available from: `http://doi.acm.org/10.1145/2684080.2684081`.

13. Zhang Y, Skolnick J. TM-align: a protein structure alignment algorithm based on the TM-score. Nucleic Acids Research. 2005;33(7):2302–2309.

14. Shindyalov IN, Bourne PE. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. Protein Engineering. 1998;11(9):739–747.

15. Krasnogor N, Pelta DA. Measuring the similarity of protein structures by means of the universal similarity metric. Bioinformatics. 2004;20(7):1015–1021. doi:10.1093/bioinformatics/bth031.

16. Sharma A, Papanikolaou A, Manolakos ES. Accelerating All-to-All Protein Structures Comparison with TMalign Using a NoC Many-Cores Processor Architecture. In: Proceedings IPDPS Workshops; 2013. p. 510–519.

17. Cock PJA, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics. 2009;25(11):1422–1423. doi:10.1093/bioinformatics/btp163.

18. Sharma A, Manolakos ES. Efficient multi-criteria protein structure comparison on modern processor architectures. BioMed Research International. 2015;vol. 2015:13 pages. doi:http://dx.doi.org/10.1155/2015/563674.

19. Zhu J, Weng Z. FAST: a novel protein structure alignment algorithm. Proteins. 2005;58(3):618–627.

20. Malod-Dognin N, Przulj N. GR-Align: fast and flexible alignment of protein 3D structures using graphlet degree similarity. Bioinformatics. 2014;30(9):1259–1265. doi:10.1093/bioinformatics/btu020.

21. Sharma A, Manolakos ES. Multi-criteria protein structure comparison and structural similarities analysis using pyMCPSC. PLOS ONE. 2018;13(10):1–15. doi:10.1371/journal.pone.0204587.