Parallel Processing Letters

GRID DEPLOYMENT OF BIOINFORMATICS APPLICATIONS: A CASE STUDY IN PROTEIN SIMILARITY DETERMINATION

MATTEO COMIN, CARLO FERRARI and CONCETTINA GUERRA *

Department of Information Engineering, University of Padova
via Gradenigo 6a, 35131 Padova, Italy

Received 21 November 2003
Revised 17 April 2004
Communicated by A. Apostolico

ABSTRACT

In this paper we present a scenario for the grid immersion of the procedures that solve the protein structural similarity determination problem. The emphasis is on the way various computational components and data resources are tied together into a workflow to be executed on a grid. The grid deployment has been organized according to the bag–of–service model: a set of different modules (with their data set) is made available to the application designers. Each module deals with a specific subproblem using a proper protein data representation. At the design level, the process of task selection produces a first general workflow that establishes which subproblems need to be solved and their temporal relations. A further refinement requires to select a procedure for each previously identified task that solves it: the choice is made among different available methods and representations. The final outcome is an instance of the workflow ready for execution on a grid. Our approach to protein structure comparison is based on a combination of indexing and dynamic programming techniques to achieve fast and reliable matching. All the components have been implemented on a grid infrastructure using Globus, and the overall tool has been tested by choosing proteins from different fold classes. The obtained results are compared against SCOP, a standard tool for the classification of known proteins.

Keywords: Protein Structure Comparison, Structural Biology, Grid Deployment, Grid Workflow Design, Distributed Systems.

1. Introduction

Grid is emerging as a novel paradigm for high performance distributed information processing. In this paper we discuss the use of a grid as a computational infrastructure for applications of structural bioinformatics. Given a rich and integrated set of software components (and their related data sets) that each solve a specific task of a general problem, we describe their deployment on a grid that results in more powerful tools of immediate use by researchers in the bioinformatics community worldwide.

Bioinformatics problems involve solutions that can exploit grid characteristics. Grid supports the aggregation of computational resources for those problems that

*email: {ciompin,carlo,guerra}@dei.unipd.it

cannot be solved on a single system. Applications in bioinformatics usually have high computational requirements. For instance, the protein docking problem involves the screening of thousands of drug candidates to determine the structure of large complex assemblies while the protein structure prediction, (that is the determination of the three-dimensional structure of a protein from its sequence) can benefit from molecular dynamics simulations procedures running on multiple supercomputers.

Bioinformatics applications are data-intensive, where new information is synthesized from data maintained in geographically distributed repositories, digital libraries and database. The Protein Data Bank (PDB) [1] [2] currently contains about 22,000 structure descriptions, while dedicated databases (like SCOP [3], FSSP [4]) are accessed for specific purposes. Within grids it is feasible to organize complex, high-volumes of data flows through multiple levels of hierarchy.

Finally, it must be observed that the field of Bioinformatics is highly collaborative and multidisciplinary, involving many fields from biology to computer science, statistics, physics and mathematics. It is not unusual for teams to span institutions, states, countries and continents. These teams may want to share, in addition to computer power and storage devices, also analysis procedures and computational results. Grid well supports collaborative computing applications, where different types of end users want to work in the same virtual space.

As computational grids enable large-scale sharing of resources and data among geographically distributed sites, the application developers must be given proper and easy-to-use protocols and services for security, resource scheduling and data management [5], [6], [7]. The grid infrastructure also includes mechanisms for discovering and negotiating access to remote resources, in order to let an application be routed to the most appropriate resource: a dedicated supercomputer if it features high degree of parallelism or one or more idle workstations if it can be distributed across several computational nodes. All these services and protocols form the grid middleware: most Grid middleware projects are being built on Globus [8] that provides many of the needed services.

The grid deployment of bioinformatics applications sees a number of projects under development. Among them there are the Asia Pacific BioGrid Initiative, the North Carolina BioGrid, the Canadian BioGrid, the EUROGRID project and the Biomedical Informatics Research Network. Their goal is to build grid infrastructures and services for resource discovery, workflow enactment and distributed query processing that are tailored to bioinformatics applications. Often such applications have less stringent storage requirements than applications in other disciplines, for instance in energy physics (typically order of Gigabytes in bioinformatics rather than Petabytes), but they are characterized by high semantic complexity and highly heterogeneous data structures. To address these issues, some projects concentrate on designing abstractions to describe and combine the various computational components of a bioinformatics application for a grid immersion [9], [10]. Other grid related projects deal with the development of a problem solving environment to build and execute bioinformatics applications as distributed workflows of software components [11].

In the following we discuss an approach to designing bioinformatics applications

running on grids and we present a possible scenario for the grid immersion of the procedures that solve an important problem in proteomics: the protein structural similarity determination. Proteomics is focussed on the study of proteins and other organic molecules, and their interactions. Analyzing proteins related biomechanisms can help in medicine and pharmacology, as well as in less conventional fields, like biomaterials design and application, agricultures and environmental studies. As proteins properties and functionalities are strictly related to their 3D shape, proper methods have to be used to analyze geometric molecular similarities to discover and understand common functions. Geometric similarities can be analyzed looking at partial or global alignment between proteins components. Hence the structural similarity determination is a key component of functional classification of known proteins and for behavior prediction of new proteins. Its solution involves many different representations, techniques and algorithms.

In this paper the emphasis is on the various computational components and data resources involved in the structural comparison problem and how they can be tied together into a workflow to be executed on a grid. The next section describes the way bioinformatics applications can be designed from a set of basic available tools and data sets and how their deployment on a grid can be organized according to the bag–of–service model. Section 3 reviews the protein structural comparison problem. Section 4 describes the basic computational tools and the way they form a complete workflow and dataflow; some validating experiments are also presented.

2. Grid immersion of bioinformatics applications

An application is immersed on a grid when a proper procedure, either automatically or manually, brings the application components to run on grid nodes and to access, share and exchange data according to the specification of the application itself. The application components can be already on their related grid nodes or they can migrate to the available grid nodes, according to a general grid scheduling policy. Functional and structural constraints due to the application, should be described in a clear manner, using, for instance, a description language for work and dataflow.

Grid services completely hide to the users all the low level details. Ideally, a user should be able to design a workflow for his specific application by selecting among the available computational modules and data repositories those that best fit the application requirements. The workflow should then guide the grid deployment of the application, without the user getting involved with grid services such as scheduling and configuration of the infrastructure.

Furthermore, the user may find that a given workflow, even though appropriate for certain proteins or classes of proteins, fails to produce meaningful results for other proteins. Thus, he may need to experiment with different methods and workflows for the same problem applied to different inputs. In addition, he may want to be able to display the outputs of the computation, either intermediate or final, using appropriate visualization tools. The modules may be available at distant locations, use different protein representations and corresponding data sets, or may need additional storage to build and use new representations. In al cases the user should be shielded from the details of the overall computation, ignoring the module

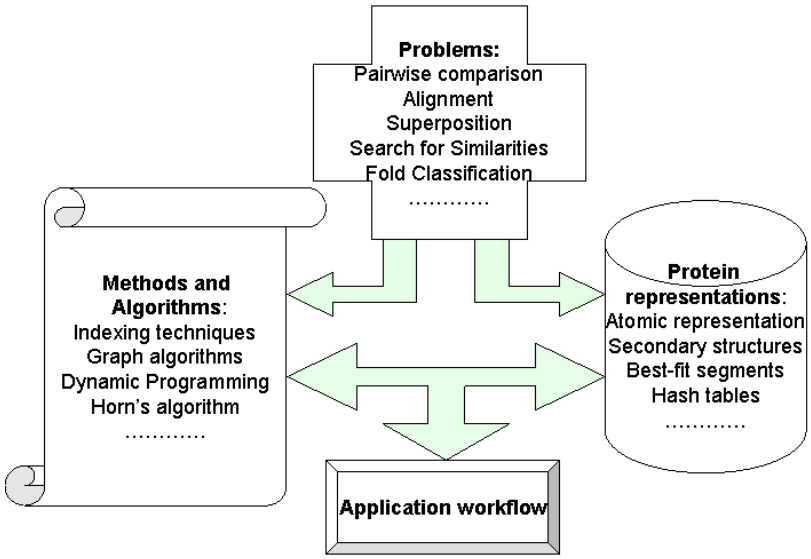

Fig. 1. Bag–of–tasks for structural bioinformatics.

locations, their computational requirements and the module interfaces.

A grid framework may facilitate this scenario by making it possible for users to specify the computational tasks and requirements allowing each time the creation of a new workflow built from the available software and storage resources. A bag–of–tasks composed by independent tasks in structural bioinformatics is made available to the grid community: the tasks can be executed in any order and need not to run simultaneously. Figure 1 shows an example of a bag of tasks with their associated computational modules and data sets that can be variously combined and interfaced to satisfy different requirements. At the design level, the process of task selection produces a general workflow that shows the temporal relation among the tasks involved (see figure 2).

The user should be assisted in the process of discovery and selection of the necessary services from a bag of computational components that solve the selected tasks. The outcome of this selection should consist of a workflow ready for execution on a grid. As an example, figure 3 represents the workflow corresponding to our approach (to be described later) to protein structural comparison.

Besides procedural aspects, a bioinformatics application design usually involves different data representations at different granularity levels for the same entities. For instance, a protein can be described either as a sequence of residues, or as set of geometrical structures (alpha helix and beta strand), or as a set of atomic coordinates of its tertiary structure and so on. Intermediate data representations and

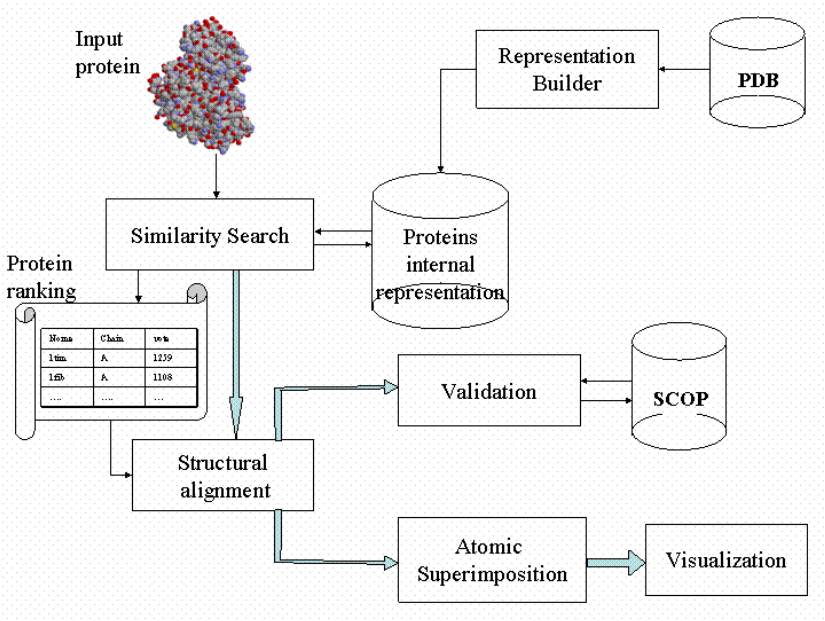

Fig. 2. A workflow for structural comparison of proteins.

results can be produced and stored in a long term fashion, to let other users check methods and procedures, and produce statistics. These data could also migrate from site to site for a better accessibility and could be updated when changes to the original data occur.

Thus, the crucial aspect is often not the data dimension but the variety of data formats and semantics. While the former problem can be addressed designing proper conversion routines, the latter one has to be solved at the design level, by implementing strategies ranging from data replica to a complete re-computation of the needed information. Hence data can be accompanied either by their metadescription or by the procedures that have produced them.

The application design follows from the careful analysis of the pros and cons of the available tools with respect to the chosen protein representations and the application performance requirements.

3. Structural similarity determination

In this section we review the structural comparison problem and describe the various components that may be combined to provide an efficient solution. There are mainly two instances of the problem that one wants to solve:

- the extraction from a data base of known proteins (the test set) of a subset of proteins with a high degree of structural similarity with a query protein.

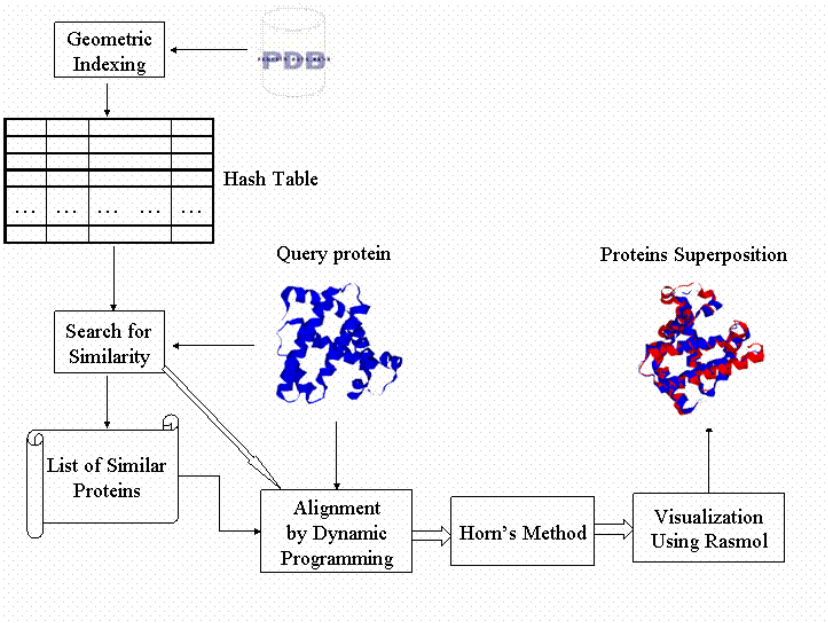

Fig. 3. An instance of workflow for protein structural comparison.

- a pair-wise comparison that typically involves the determination of the optimal alignment of two given proteins,

These two instances of the comparison are obviously related and can be solved by sharing a few computational methods.

Structural alignment is a complex problem (NP-hard) for which many heuristics have been proposed in the literature. There are a variety of computational methods involved in the solution of the two instances of the comparison problem as well as a few web servers where to submit request for similarity searching.

Among the approaches used for pair-wise comparison, the ones that proved most successful are dynamic programming techniques [12], [13], [14], graph algorithms [15], [16], [17], [18], and methods based on the optimization of a suitable metric relating distance matrices [19], [20]. For the one-to-all and all-to-all comparison, indexing techniques have been the most popular ones due their ability to store protein representations that allow fast retrieval of similarity information [21]. An important distinction among methods is whether the comparison is based on the atomic representation of the proteins or on the secondary structure representation. It is often the case that the one-to-all comparison uses an initially representation based on secondary structures and moves to the atomic representation for the more refined pair-wise comparison and superposition.

Among the most popular web servers there are: DALI [4], CE [22], SSAP [23] and VAST [24]. It is beyond the scope of this paper to review in detail all approaches.

Good surveys exists on the subject [25], [26], [27], [28].

Protein structural similarity determination can be approached both at a coarse grain level of details, studying skeletal protein properties, as well as at a fine grain level of details considering either atoms or protein surface points.

All the pair-wise protein comparison approaches proposed in the literature share a common structure composed by three main steps: protein representation determination, similarity evaluation and alignment determination. Selecting and computing the most suited protein representation is an important preprocessing step that affects the way similarity algorithms are designed. An algorithm for determining the structural similarity produces a numerical evaluation of the goodness of the obtained solution. Typically, such an algorithm also generates an alignment of the two proteins, but sometimes the alignment is determined by a separate procedure. Finally, it is extremely useful to compute and visualize the superposition of the two proteins at the atomic level.

Given a protein (the query protein) and a set of other proteins (the test set), the one-to-all protein comparison generally iterates the pair-wise comparison over all proteins of the test set. To optimize the overall method, it is possible to devise a schema consisting of four steps: representation determination, similarity evaluation, protein ranking, alignment computation. Thus the two instances of the comparison problem can be implemented by sharing a few computational modules. In both instances, the first step is related to find the best suited protein representation. A gross filtering step is useful in one-to-all comparison for discarding from the test set those elements that are clearly unrelated to the query protein. In other words, this step extracts from the test set those proteins that are likely candidates for matching and that form the candidate set. The structural similarity evaluation and alignment of the query protein with each of the proteins in the candidate set is then performed and the significance of the results is evaluated. Finally, it is required to compute and display the atomic superposition of the query protein with some selected candidates.

Since structural similarity can be inferred from sequence similarity, often one more processing step is added to the above to check for sequence similarity thus reducing the number of proteins that need to be examined. This step can be performed as the first step of the comparison process to immediately eliminate from the data set of proteins all sequence homologs.

4. Our approach to protein similarity determination for grid execution

Our approach to one-to-all protein structure comparison is based on a combination of indexing and dynamic programming techniques to achieve fast and reliable matching. Indexing uses the secondary structure representation of proteins and produces an initial solution made of a set of proteins that are good candidates for matching. This set is to be refined at a later stage by iteratively performing pairwise comparison of the query protein with each of the candidate proteins.

Indexing consists of extracting from all proteins in the data set relevant geometric properties of their 3D structure and using them to access a look-up table (hash table). This table is built in a preprocessing stage, and, once compiled, is used to retrieve candidate proteins in an efficient way. Pairwise comparison is based on

dynamic programming followed by the atomic protein superposition using Horn's algorithm. The overall approach is summarized by the workflow of figure 3.

4.1. Indexing

Geometric indexing techniques, initially proposed in the field of computer vision and referred to as geometric hashing, allow fast recognition by moving the complexity of computation to the preprocessing that computes and stores redundant model representations. In the field of proteomics, indexing has been used by several approaches to protein structure comparison, applied either to the atomic representation of the protein structure or to its secondary structure representation [19], [21], [29]. The approaches differ in many aspects, notably the choice of the geometric property used to build and access the hash table, the way the table is organized and searched and the metric used to rank the obtained hypotheses.

Our strategy looks for associations between triplets of secondary structures of the query protein Q and triplets of secondary structures of the stored proteins. Each secondary structure is modeled as a line segment: the segment associated to an helix is the axis of the helix, for the strand it is the segment that best fits the set of C_α atoms belonging to the strand. Let (q_i, q_j, q_k) be a triplet of secondary structures segments of Q with $i < j < k$ so that the three secondary structures are in increasing order along the sequence. Let α_{ij}, α_{ik}, and α_{jk} be the three angles and d_{ij}, d_{ik}, d_{jk} be the three distances formed by the segments of the triplet. The distance is measured as the Euclidean distance between the midpoints of two segments. Also, let P be a stored protein and (p_s, p_t, p_u) a triplet of secondary structures of P with angles $\phi_{st}, \phi_{su}, \phi_{tu}$ and distances h_{st}, h_{su}, h_{tu} The two triplets of secondary structures are considered equivalent if each difference $|\alpha_{ij} - \phi_{st}|$, $|\alpha_{ik} - \phi_{su}|$ and $|\alpha_{jk} - \phi_{tu}|$ is less than a given threshold TA (10 degree in our implementation) and each difference $|d_{ij} - h_{st}|$, $|d_{ik} - h_{su}|$ and $|d_{jk} - h_{tu}|$ is less than the threshold TD.

We build a three-dimensional hash table that allows to retrieve equivalent triplets of elements very efficiently. The table stores all triplets of segments of secondary structures of all proteins in the data set. For each triplet the values of the three angles are discretized into uniform intervals and used as indexes to access a cell of the table. Each table cell contains a list of records, one for each triplet that hashed into it; a record stores the name of the protein containing the triplet, the identifiers of each of the three secondary structures and the three distances between the three segments associated to the secondary structures. More details on our the indexing scheme used for the hash table can be found in [30].

The construction of the table is the most critical part of the overall processing. First, the insertion into the table of a single protein with n secondary structures requires $O(n^3)$ execution time. Second, as new proteins are inserted, the table becomes very large thus making accesses to the linked list of cells in secondary memory very costly. With the current size of the PDB (more than 22,000 proteins and an average number of secondary structures of about 15) the size of the table after the insertion of about 20,000 proteins from the PDB is above 10 Gigabytes. Notice that some proteins could not inserted due to inaccuracies in the PDB data. The table construction would take several hours or perhaps days of computing time on

a standard PC or workstation. For these reasons, we have resorted to a distributed implementation of the table construction phase on the grid using MPICH built on top of Globus [8].

The distributed implementation of the algorithm was described in an earlier paper [31] and only a brief sketch is given here. The main goals are to achieve a good load balancing and to minimize secondary memory accesses. They are complicated by the highly non uniform distribution of the entries of the table, as described in [32]. Another complication arises from the large difference in protein size, ranging from 3 secondary structures (we did not consider proteins with fewer that 3 secondary structures) to hundreds of them. Furthermore, the data set is subject to frequent updates corresponding to the new releases in the Protein Data Bank.

The table is built and partitioned across the nodes of the computational Grid. A given node acts as a master and is in charge of distributing and collecting all information to and from the slave nodes. During the table construction phase, the master repeatedly sends separate blocks of input proteins to the slaves and each slave builds and maintains its own sub-table. Due to its large size, each sub-table is kept in secondary memory. It is stored as a doubly-linked list of elements: there is an element in the list associated to each cell of the three-dimensional table with two pointers: one to the next cell in the table and the second to a list of records, one record for each triplet of secondary structures that hashed into the cell.

Each slave completes the insertion of the assigned list of proteins into its sub-table, signals its availability to the master and then waits for a new block of proteins. This process is repeated until all proteins have been inserted. Despite its simplicity, the proposed approach to table partition works quite well achieving a good load balancing. In fact, using all the PDB proteins as inputs we have obtained a sub-table occupancy very close to the optimal: the difference in sub-table size in all nodes is less than 2% of the size of the sub-tables. The overall execution time is about 14hours on 4 nodes (in our current implementation, we use 4 standard PC).

4.1.1. Ranking similarity hypotheses

Given a query protein Q, the proteins that are structurally similar to Q are extracted from the hash table of the target proteins and are ranked according to a given similarity measure. The similarity measure between two proteins Q and P is expressed as the number of equivalent triplets of Q and P.

For retrieval of similarities, a procedure similar to that used for inserting a protein in the hash table is used. That is, for each triplet (q_i, q_j, q_k) of segments associated to the secondary structures of the query protein Q their three angles are computed and used as indexes to a table cell. All triplets of the target proteins equivalent to (q_i, q_j, q_k) are stored in that same cell. For each protein containing one such triplet a counter is incremented. After all triplets of segments of Q have been considered, the target proteins are sorted according to the value of their counters.

The distributed implementation of the above procedure on the nodes of a grid follows the same simple master-slave paradigm as described for the table construction. The master node is in charge of broadcasting to all the slaves the information about the query protein. Each slave performs the search on its sub-table, then

sends its partial ranking back to the master that produces the final ranking. The experiments conducted on several query proteins against a database of about 20,000 stored proteins have shown that this search is really fast for most proteins: typically it takes few minutes and for proteins of small size only few seconds.

The indexing method discussed in this subsection returns a ranking of candidate proteins but does not generate an alignment of secondary structures of the query protein with each of the candidate proteins. The method, however, provides a good basis for a more refined analysis that may be applied to few candidate proteins only, those highly ranked by the indexing method.

An alignment procedure based on dynamic programming is briefly described in the next subsection.

4.2. Secondary Structure Alignment

Consider protein Q with n secondary structures $(q_1, \cdots, q_n)$ and a candidate protein P with m secondary structures $(p_1, \cdots, p_m)$. Our structural alignment procedure produces an alignment that can be described as an ordered set of pairs (i, j) associating secondary structure q_i of Q to secondary structure p_j of P. The alignment satisfies the continuity constraints, i.e. if (i, j) and (h, k) are two pairs in the set and $i < h$ then $j < k$. The method allows gaps in the alignment that correspond to non sequential i values in consecutive pairs.

The alignment procedure is based on the notion of equivalent triplets of secondary structures introduced in the previous subsection. Given two equivalent triplets (q_{i1}, q_{i2}, q_{i3}) of Q and (p_{j1}, p_{j2}, p_{j3}) of P there are three possible pairs (p_{i1}, q_{j1}), (q_{i2}, p_{j2}), and (q_{i3}, p_{j3}) associating segments of Q to segments of P. The method defines a score for each association (q_i, p_j) that is the number of times the pair appears in all equivalent triplets of Q and P. The alignment procedure optimizes the following function: the sum over all secondary structures associations of the above score, i.e. the number of times each association is present in any two equivalent triplets of Q and P. The scores are stored in a two-dimensional matrix D, where the entry $D(i, j)$ corresponds to the pair of segments q_i of Q and p_j of P. A dynamic programming algorithm determines an optimal non decreasing path in the matrix D. Details of the alignment algorithm will be found in a forthcoming paper. Other scoring functions that penalize gaps in the alignment and introduce factors related to the compactness of the solution are currently being investigated.

4.3. Protein Superimposition

The final stage of the proposed scenario for protein structure comparison is the superposition of the two molecules, that is the determination of the rigid transformation that results in the "best" overlap of the two proteins. At this stage we are taking into consideration the atomic representation of the proteins. The rigid transformation is defined as the one that minimizes the Root Mean Square Deviation (RMSD) between pairs of corresponding points of the two proteins. Given two sets of corresponding points, the problem of minimizing the RMSD between the points has a closed form solution. There a number of different approaches to solve this minimization problem. We choose Horn's algorithm [33] that provides a fast and

efficient solution. Horn's algorithm computes, in time proportional to the number of points, a 4×4 transformation matrix that minimizes the RMSD between the points.

In our approach, we determine a transformation based on the points belonging to the secondary structures segments aligned by the dynamic programming algorithm described in the previous subsection. The dynamic programming algorithm produces as result a list of associations between secondary structures; such correspondences are promising in terms of global superposition thus we take this information into account. In the following we describe how to select pairs of corresponding points of the aligned structures as input to Horn's procedure.

The three main steps of the superposition procedure are:

1. Compute an initial rigid transformation based on the starting and ending residues of the aligned secondary structures.

2. After applying the obtained transformation to one of the proteins, find pairs of nearest atoms for each pair of aligned secondary structures.

3. Compute the transformation that minimizes the RMSD between the obtained pairs of atoms.

4. Repeat steps 2 and 3 until RMSD converges.

The number of points involved in this step is typically very small leading often to a solution of low quality. To improve it, we need to refine our choice of corresponding points, based on the results of this first step. To this end, we select pairs of nearest atoms from corresponding secondary structures, that is pairs such that each atom of the pair is the nearest of the other and they are at distance less than a given threshold.

The application of the algorithm to the new set of points produces a new transformation. The overall procedure iterates through steps 2 and 3 to adjust the transformation until the RMSD converges, and hopefully the number of atoms increases. The final result is a roto-translation that superimposes the two proteins, aligning the most promising secondary structures.

4.4. Validation

To test our method we devised a set of experiments by choosing proteins from different fold classes and running the different modules of the comparison task either in isolation or combined to solve the one-to-all comparison problem. We compare our results against the SCOP [34] that provides a classification of known protein structures based on visual inspection as well as on automatic techniques. We are interested in verifying if SCOP classifies our top ranked proteins as belonging to the same fold as the query protein. In our method the final results are in the form of a list of items, each item refers to a single chain of a protein. The list is in descending order, according to an associated measure of similarity. An item of the list is considered to be *correct* if it is in the same fold of the query item, according to SCOP.

Starting from a subset formed by the first n items of the list we build subsets of increasing sizes by repeatedly inserting consecutive proteins from the list with a lower similarity measure. The assessment process is based on the idea of counting how many proteins of each such subset can be considered *correct*. An accuracy measure can be associated to each subset, given by the ratio between the number of *correct* items over the total size of the subset.

Test Set dimension	Correct items	Accuracy %
100	100	100,00%
200	200	100,00%
300	300	100,00%
400	400	100,00%
500	498	99,60%
600	596	99,33%
700	685	97,86%
800	769	96,13%
900	831	92,33%
1000	888	88,80%
1100	937	85,18%
1200	970	80,83%
1500	1072	71,47%
2000	1216	60,80%

Table 1: Accuracy of the comparison results for protein 1TIM chain A.

As an example, Table 1 summarizes the results for protein 1TIM chain A that is classified as the fold TIM alpha/beta-barrel by SCOP. In the table, the top ranked proteins are grouped into sets whose size increases in steps of 100. As it can be seen, the accuracy is the maximum possible for the top ranked 400 items, all of which are classified by SCOP as TIM alpha/beta barrels; then it decreases very slowly and at 1000 it is still 88% .

5. Conclusions and Future Work

We have presented an application of structural bioinformatics and discussed its grid immersion. The application can be solved by a combination of various components each with different computational and data requirements. We have provided software modules for each of the above components including the visualization of the results of the alignment. All these components have been implemented on a grid infrastructure using Globus. In particular, the indexing technique for fast similarity search has been distributed among the nodes of a grid using MPICH built on Globus. All the components can now be used in isolation, their combined use can only be achieved through a careful examination of the component interfaces. Our future work will consist of providing services to assist a grid user in the process of combining the modules into a workflow for execution on a grid.

Acknowledgements

Support for Guerra and Ferrari was provided in part by the Italian Ministry of University and Research under the FIRB Project "Enabling Platforms for High Performance Computational Grids in Scalable Virtual Organizations", and by the European DataGrid project.

References

[1] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, **28** (2000) 235–242.
[2] http://www.rcsb.org/pdb/
[3] http://scop.mrc-lmb.cam.ac.uk/scop/
[4] hhtp://www.ebi.ac.uk/dali
[5] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications,* **23** (2001) 187–200.
[6] I. Foster, C. Kesselman eds. *The Grid: Blueprint for a New Computing Infrastructure*, (Morgan Kaufmann Publisher, San Francisco, 1998).
[7] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, **15**, (2001).
[8] http://www.globus.org
[9] M. Addis, F. Ferris, M. Greenwood, D. Marvin, P. Li, T. Oinn, A. Wipat. Experiences with e-Science workflow specification and enactment in bioinformatics (2001).
[10] W. Cirne, K. Marzullo. Open Grid: A user-centric approach to grid computing, *Proc. of the 13th Symposium on Computer Architecture and High Performance Computing*, 2001.
[11] M. Cannataro, D. Talia. Knowledge grid: an architecture for distributed knowledge discovery. *CACM*, **46** (2003) 89–93.
[12] M. Gerstein, M. Levitt. Comprehensive Assessment of automatic structural alignment against a manual standard, the scop classification of proteins. *Protein Science*, **7** (1998) 445–456.
[13] A.P. Singh, D.L. Brutlag. Hierarchical protein structure superposition using both secondary structures and atomic representations. *Proc. Fifth Int. Conf. on Intell. Sys. for Mol. Biol.*, Menlo Park, (1997) 284–293.
[14] I.N. Shindyalov, P.E. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Eng.*, **11** (1998) 739–747.
[15] R.A. Abagyan, V. N. Maiorov. An Automatic Search for Similar Spatial Arrangements of α helices and β-strands in globular proteins. *Journal of Molecular Structural Dynamics* **6** (1989) 1045–1060.
[16] N.P. Brown, C.A. Orengo, W.R. Taylor. A protein structure comparison methodology. *Comp. Chem.* **27** (1996) 359-380.
[17] G. Lancia, R. Carr, B. Walenz, S. Istrail. Optimal PDB structure alignments: A Branch-and-Cut algorithm for the maximum contact map overlap problem. *Proc. 5th ACM REsearch in COMputational Biology* (2001) 193–202.
[18] W.R. Taylor, C.A. Orengo. Protein structure alignment. *J. Molec. Biology*, **208** (1989) 1–22.
[19] L. Holm, C. Sander. Mapping the protein universe. *Science*, **273** (1996) 595–602.
[20] L. Holm, C. Sander. The FSSP database: fold classification based on structure-structure

alignment of proteins. *Nucleid Acids Research*, **24(1)** (1996) 206–209.
[21] D. Fischer, C.J. Tsai, R. Nussinov, H. Wolfson. A 3D sequence-independent representation of the protein data bank. *Protein Engineering*, **8** (1995) 981–997.
[22] http://cl.sdc.ede/ce.html
[23] http://www.biochem.ucl.ac.uk/orengo/ssap.html
[24] http://www.ncbi.nlm.nih.gov/Structure/VAST/vast.html
[25] P.E. Bourne, H. Weisseig eds.. *Structural Bionformatics*, (Wiley, Ney York, 2003).
[26] C. Ferrari, C. Guerra. Geometric methods for protein structure comparison. *Protein Structure Analysis and Design* , (C. Guerra, S. Istrail, eds.), Lecture Notes in Bionformatics (Springer-Verlag, Berlin, 2003).
[27] G. Lancia, S. Istrail. Protein structure comparison: algorithms and applications. *Protein Structure Analysis and Design* , (C. Guerra, S. Istrail, eds.), Lecture Notes in Bionformatics, (Springer-Verlag, Berlin, 2003).
[28] C. Lemmen, T. Lengauer. Computational methods for the structural alignment of molecules. *J. of Computer-Aided Molecular Design*, **14** (2000) 215–232.
[29] A. Alesker, R. Nussinov, H.J. Wolfson. Detection of non-topological motifs in protein structures. *Protein Engineering*, **9** (1996) 1103–1119.
[30] C. Guerra, S. Lonardi, G. Zanotti. Analysis of proteins secondary structures using indexing techniques. *IEEE Proc First Int. Symposium on 3D Data Processing Visualization and Transmission*, (2002) 812–821.
[31] C. Ferrari, C. Guerra, G. Zanotti. A Grid-aware approach to protein structure comparison. *Journal of Parallel and Distributed Computing*, Special Issue on "High Performance Computational Biology" **63** (2003) 728–737.
[32] D.E. Platt, C. Guerra, I. Rigoutos, G. Zanotti. Global secondary structure packing angle bias in proteins. *Proteins: Structure, Function, and Genetics*, (in press).
[33] B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am.*, **4(4)** (1987) 629–642.
[34] A.G. Murzin, S.E. Brenner, T. Hubbard, C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Molecular Biology*, **247** (1995) 536–540.