Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

INFORMATICA

Ciclo XXVII

Settore Concorsuale di afferenza: 01/B1 - INFORMATICA

Settore Scientifico disciplinare: INF/01 - INFORMATICA

GRAPH ALGORITHMS FOR BIOINFORMATICS

Presentata da: **GIUSEPPE PROFITI**

Coordinatore Dottorato	**Relatore**
Prof. Paolo Ciaccia	**Prof. Rita Casadio**

Esame finale anno 2015

Abstract

Biological data are inherently interconnected: protein sequences are connected to their annotations, the annotations are structured into ontologies, and so on. While protein-protein interactions are already represented by graphs, in this work I am presenting how a graph structure can be used to enrich the annotation of protein sequences thanks to algorithms that analyze the graph topology. We also describe a novel solution to restrict the data generation needed for building such a graph, thanks to constraints on the data and dynamic programming. The proposed algorithm ideally improves the generation time by a factor of 5. The graph representation is then exploited to build a comprehensive database, thanks to the rising technology of graph databases. While graph databases are widely used for other kind of data, from Twitter tweets to recommendation systems, their application to bioinformatics is new. A graph database is proposed, with a structure that can be easily expanded and queried.

Acknowledgements

During the past three years many people helped me with guidance, technical and scientific insights or just some hints. I would like to thank:

- My advisor, prof. Rita Casadio, for letting me experiment with different ideas and mostly for showing me the uttermost importance of a solid understanding of the data and the statistics behind them while performing research;

- My supervisor, prof Luciano Margara, and the PhD board of the Computer Science and Engineering department for trusting my ability to complete such a demanding program;

- All the members of the Bologna Biocomputing Group, in random order: Pier Luigi Martelli, Piero Fariselli, Damiano Piovesan, Ivan Rossi, Castrense Savojardo and Francesco Aggazio; for the helpful insights and discussions;

- My fellow PhD colleagues, for the topics outside my main focus we discussed.

And, of course, my family, for reminding me that there is life outside the academia.

Dedication

To my parents, who worked hard to let me grow my interest and knowledge in mathematics and computer science, in spite of the hostile environment.

To Erica: be it by chance or by design, here we are now, after a long chain of causes and effects.

‘The paths fork and divide. With each step you take through Destiny’s garden, you make a choice; and every choice determines future paths. However, at the end of a lifetime of walking you might look back, and see only one path stretching out behind you; or look ahead, and see only darkness’

The Sandman, by Neil Gaiman

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Objectives

The fields of computational biology and bioinformatics are quite popular. The amount of data generated by different projects in these research fields is huge and finding new ways to manage it is challenging. Most of the data generated can be connected to other similar data. In this way, new insights on the properties of biological systems can be obtained, like effects of genomic mutations on the development of diseases.

The application of graph theory to such fields could then be useful, provided that the domain is modelled properly and that smart and fast technologies are used to manage the data and to execute graph theoretic algorithms.

The goals of this work aim to demonstrate how such approach can be applied to the problem of functional annotation of protein sequences. One goal is to provide a scalable framework to tackle the data size problem, another one is to enhance existing annotation methods exploiting graph algorithms. The third and last objective is to design an easy to use framework for managing heterogeneous data about biological sequences and their annotation, highlighting those emerging from the proposed algorithms.

1.2 Contributions

The author tested different algorithms on graphs for enhancing the annotation of biological sequences. This is detailed in chapter 3.

The generation of the raw data behind the algorithms was also taken into account. First, new data was generated with the technical support of Daniele Cesini of the INFN-CNAF centre, which manages the Italian Grid Infrastructure. The experience on the grid infrastructure highlighted the need of some improvements, that have been designed and are presented in chapter 4. Both the design and implementation of the algorithms were carried on by the author. Some optimization on the algorithm were proposed by Piero Fariselli. A paper on this subject is going to submitted to an international peer-reviewed journal.

To manage the interconnected data, various solutions have been tested, leading to the final proposal presented in chapter 5. The deployment of the proposed framework, the implementation of the helper programs and the actual population of the database have been performed by the author.

1.3 Statement of Originality

To the author's knowledge, the combination of methodologies presented in this work is new. Some of the techniques presented in chapter 3 were applied previously, but to smaller and more bounded domains, or to obtain different insights on the biological data.

The solutions presented in chapter 4 are specific for the method that was improved by this work.

The use of the framework proposed in chapter 5 in this context is new, given also that the technology used is still being investigated for the field of bioinformatics [34]. The only other project using a similar approach is Bio4j [47], that was developed at the same time as the work presented in this thesis.

1.4 Publications

- [52] Damiano Piovesan, Giuseppe Profiti, Pier Luigi Martelli, and Rita Casadio. The human "magnesome": detecting magnesium binding sites on human proteins. *BMC Bioinformatics*, 13(Suppl 14):S10, 2012

- [50] Damiano Piovesan, Pier Luigi Martelli, Piero Fariselli, Giuseppe Profiti, Andrea Zauli, Ivan Rossi, and Rita Casadio. How to inherit statistically validated annotation within BAR+ protein clusters. *BMC Bioinformatics*, 14(Suppl 3):S4, 2013

- [56] Giuseppe Profiti, Damiano Piovesan, Pier Luigi Martelli, Piero Fariselli, and Rita Casadio. Community detection within clusters helps large scale protein annotation - preliminary results of modularity maximization for the bar+ database. In *4th International Conference on Bioinformatics Models, Methods and Algorithms*, pages 328–332, 2013

- [57] Giuseppe Profiti, Damiano Piovesan, Pier Luigi Martelli, Piero Fariselli, and Rita Casadio. Protein sequence annotation by means of community detection. In *IWBBIO 2013*, pages 753–755, 2013

- [54] Damiano Piovesan, Giuseppe Profiti, Pier Luigi Martelli, Piero Fariselli, Luca Fontanesi, and Rita Casadio. SUS-BAR: a database of pig proteins with statistically validated structural and functional annotation. *Database*, 2013:bat065, 2013

- [53] Damiano Piovesan, Giuseppe Profiti, Pier Luigi Martelli, Piero Fariselli, and Rita Casadio. Extended and robust protein sequence annotation over conservative non hierarchical clusters: the case study of the ABC transporters. *Emerging Technologies in Computing Systems, ACM Journal on*, 9(4):27:1–27:8, November 2013

- [55] Giuseppe Profiti, Pier Luigi Martelli, and Rita Casadio. A pipeline for predicting the function of the AFP/CAFA 2014 targets at the Bologna Biocomputing Group. In *22nd Annual Conference on Intelligent Systems for Molecular Biology Automated Function Prediction Special Interest Group*, 2014

- [58] Giuseppe Profiti, Damiano Piovesan, Pier Luigi Martelli, Piero Fariselli, and Rita Casadio. Protein sequence annotation by means of community detection. *Current Bioinformatics*, 10(2):to appear, 2015

It should be noted that in [54], the first two authors should be regarded as joint First Authors.

Chapter 2

Background Theory

2.1 Introduction

The amount of biological data available is increasing at a very fast rate. Thanks to the improvement of sequencing technologies, the volume of publicly available biosequences increases with a rate higher than Moore's Law. If we consider the UniprotKB database, the world most known repository of protein sequences, this number has significantly increased in the last 5 years (see figure 2.1).

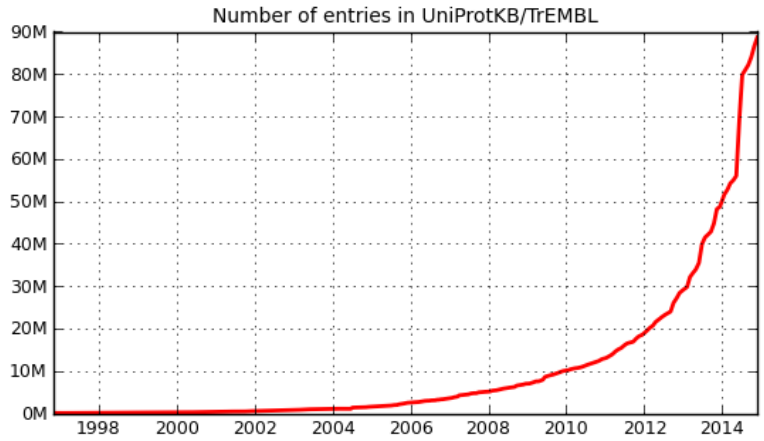

Figure 2.1: Growth of the UniprotKB database. Courtesy of European Bioinformatics Institute.[2]

[0]For more statistics, see please https://www.ebi.ac.uk/uniprot/TrEMBLstats

The quick accumulation of biological data spawns different problems: algorithms and machines need to be faster in processing these data, storage space is always in shortage and connection speed is critical. Algorithm improvement depends on the goal of the analysis, storage is cheap but not as sequencing and transmission of huge amounts of data is almost always performed in the physical world by using portable memories.

Not only information technology is falling behind in this race, but also the scientists assessing protein functions. This task is performed mostly by crystallographers, who need to scan a crystal containing the folded protein, either using x-rays or nuclear magnetic resonance spectroscopy. The goal is to identify the spatial position of the elements composing the protein, in order to better understand its functions and possible interactions.

2.2 Sequence annotation

A protein can be represented by a string of characters, each one representing a molecule called residue, or by a structure of atoms arranged in a three-dimensional space. The first is called protein sequence, the second protein structure. The main dogma in biology is that functions follows from the structure and structure follows from sequence.

The main source of information about the function of a protein, and thus about the properties conveyed by its sequence, is the analysis of its 3D structure. X-ray crystallography, nuclear magnetic resonance spectroscopy or other techniques are used to pinpoint the position of different atoms composing the protein structure.

Another source of information is the review of the literature: published studies that verified a specific function for the same protein are compared and assessed by a human expert. The confidence of this expert on the quality of the evaluated works may lead to a confirmation of the published function.

Electronic and automated annotation techniques are quite diverse in their methodologies. They may start from the sequence, comparing it to other known protein sequences annotated using

the above-mentioned methods. Patterns and motifs in the sequence may be identified, or machine learning techniques can be used to create a predictor. Other strategies rely on the known 3D structures, using for example Hidden Markov Models.

The functional annotation of sequences requires a well defined and commonly acknowledged set of keywords. Whatever the technique used, all the methods need to agree on a controlled vocabulary of terms, with a well defined semantic, to be used for annotation purposes. All the scientists experimentally verifying the structure and function of a protein should agree on the usage of such a dictionary and they should have a choice among appropriate levels of detail, given that not all experiments may lead to a fine grained assessment of the protein function. If the protein is involved in some biological process, that process needs to be clearly defined and with a unique identifier. The same applies for the function of the protein, like transporting specific molecules or being a structure in a cell or tissue.

The vocabulary may be quite specific, like the Enzyme Commission numbering system that classifies enzymes using a four number classification scheme similar to an IP address [65]. Other classification systems derive information from techniques identifying conserved domains, i.e. structures in the protein that can evolve and then differentiate the function of the protein. This kind of approach is for example used in the Pfam classification [28] that groups proteins into families. Another common annotation system is the Gene Ontology.

2.3 Gene Ontology

The most used vocabulary of terms for functional protein annotation is the Gene Ontology [4]. The terms in this ontology, usually shortened as GO terms, are layered in a direct acyclic graph of relationships. Each term is associated to a unique identifier, a name and a description. The identifier is a string starting with "GO:" and then followed by seven digits. The name is simple and easy to understand for humans, while the description helps in understanding the appropriate area of application of the term and its level of detail.

At the root of the ontology there are three terms: Biological Process (GO:0008150), Molecular

Function (GO:0005554) and Cellular Component (GO:0008372), often referenced as branches of Gene Ontology or as sub-ontologies. The Biological Process branch is about the functioning of living units, i.e. a single cell, a tissue, an organ or an organism, involving many different molecular functions. Examples of a biological process are the secretion of hormones (GO:0046879) and the response to a stimulus (GO:0050896). A term in the Molecular Function sub-ontology refers to a simple activity, like "binding" (GO:0005488) or "receptor activity" (GO:0004872). A Cellular Component term locates where the protein is usually found, be it a membrane (GO:0016020) or an extra-cellular region (GO:0005576).

The relationships between terms in the ontology are:

- "is a" represents a direct relation from a specific term to a more generic one
- "part of" is used to layer the different locations inside and outside the cell
- "has part" is the same as the previous, but in the opposite direction
- "regulates","positively regulates","negatively regulates" : when a process affects another one

The direct acyclic graph allows the traversal of the three branches from the generic root to more specific terms and vice-versa. Usually, if term A is a term B, A is called child of B and B is called parent of A. Transitivity is well defined for all the relationships and for combination of them. When annotating a protein, it is then sufficient to set the appropriate most specific term, while all its ancestors can be inferred from the ontology.

2.3.1 Distance between GO terms

The distance between two terms in the Gene ontology is a key concept needed to assess the goodness of an electronic annotation while comparing it to an experimental one. Of course, the deeper the assigned term is, the more useful for a user it will be. However, since the Gene Ontology is a direct acyclic graph, the "depth" of a term may be a different value depending on the number of hops in the specific path selected to reach the root of the branch.

There are different ways to measure the distance between two terms, and in general different measures can be converted into a score of semantic similarity [49] between them. The similarity can be evaluated taking into account either the edges in the direct acyclic graph or the properties of the nodes.

In the case of edge-based methods, one could simply count the number of edges that need to be crossed to reach a node by starting from the other one. Another technique requires to calculate the lowest common ancestor of the two terms and then to evaluate the distance of this ancestor to the root node.

The basic assumption of edge-based approaches is that nodes at the same distance from the root convey the same level of information. However this is not true in general, since branches of the ontology may be heavily populated with terms at many different depths, generating fine grained classifications, while other branches may be shallow but with a similar level of information in the leaves.

Node-based approaches starts from the concept of Information Conveyed (IC), that can be simplified as the probability of encountering the specific term in a well defined set of annotations, i.e. UniprotKB [25] or other biological databases. This system can be mixed with the common ancestor one. One example of this is the most informative common ancestor (MICA), that selects the common ancestor having the highest IC value.

Like the edge-based approaches, also this one has its drawbacks. Using probabilities of encountering an annotation is skewed towards the most popular area of research, since there are more terms from that area assigned in the knowledge-base, while less explored terms are left behind.

2.3.2 Assessment of predicted annotations

After automatically assigning annotations, it would be useful to have a metric assessing the degree of confidence on that annotation system. The results of the prediction method can be checked against already known manual annotations, like when using a test dataset in machine learning, or old predictions can be verified after manual annotations are added to repositories.

This second case is the methodology at the basis of the Challenge for Automated Function Prediction (CAFA) [59]. In this competition, the organizers release a dataset of protein sequences, asking for predictions on their future annotations. After several months, the subset of protein sequences having received a manual annotation is selected and the submitted predictions are compared against the actual set of annotations.

Methods are then evaluated using precision and recall metrics. Precision measures the ratio of correctly predicted annotations over the total of predicted annotations. Recall measures the percentage of correctly assigned annotations over the total of those assigned manually.

Other measures have been proposed, exploiting the distribution of the annotations in the knowledge-base, as in the methods discussed in section 2.3.1. One of these measures [20] exploits the probability of concurrently having both a parent and a child annotation in the corpus, and then using that joint probability distribution to compare the predicted sub-graph of the ontology to the manually assigned one.

2.4 BAR+

Among the methods for protein function prediction, the one analyzed and expanded by the research work presented in this dissertation is the Bologna Annotation Resource Plus (BAR+), an annotation method that was developed by the Bologna Biocomputing Group at the University of Bologna.

The groundwork for the annotation system is the clusterization of similar sequences and the statistical validation of their annotations. A first implementation of the method was presented in [5] as BAR, while BAR+ is a more recent and annotation rich version [51].

BAR+ is also a non-hierarchical clustering method of a large-scale set of genomic data. The clustering procedure has a very strict grouping metric, allowing the transfer of features among members of a cluster.

As explained in the previous section, the basic notion is that sequences with many identi-

cal residues should share structure and functions, then the known functions of one could be transferred to the other. In BAR+, sequences are clustered using the following constraints: percentage of identical residues, called sequence identity[3] should be equal or higher than 40% on at least 90% of the pairwise alignment length. Alignment length is given by the sum of the lengths of the two sequences, minus the length of the overlapping regions, as shown in figure 2.2. Coverage is the ratio of the length of the overlapping part of the sequences over the alignment length.

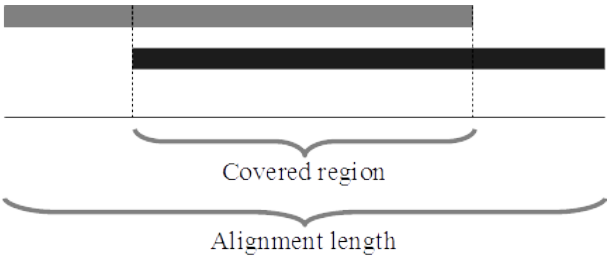

Figure 2.2: Coverage is the ratio between the covered region and the alignment length.

Sequence identity and alignment length and are obtained from an all-against-all BLAST alignment [41] of the protein sequences known at the time. BLAST uses an algorithm similar to the Smith-Waterman algorithm, a local alignment method. Both algorithms rely on a dynamic programming approach that tries to locally match two strings using a matrix of similarity scores. To speed up the computation of such alignments, BLAST was run on a Grid environment.

After obtaining the alignments, only those pairs of sequences passing the two above-mentioned constraints are considered. The selected alignments are converted into an undirected graph: vertices represent protein sequences, while edges are the similarity relations induced by the alignments. Every edge is weighted using the sequence identity (SI) score obtained from BLAST. Since the same couple of sequences may obtain a different SI score, due to the heuristic nature of the algorithm, the weight kept is the one relative to the alignment with the best coverage ratio.

Each connected component in the graph is labelled as a cluster. Each cluster inherits the annotations of the sequences belonging to it. The annotations considered are GO terms, PDB

[3]identity refers to the residues in the sequence, i.e. the letters in the string. It should not be confused with identity in the mathematical sense.

structures and Pfam families. GO terms and Pfam families of a specific cluster are then validated against the other clusters by computing a P-value. The P-value was computed applying a Bonferroni correction, leading to a more conservative validation. The P-values considered in the statistical validation are only those below 0.01.

All the validated annotations of a cluster are then transferred to all the sequences belonging to it, leading to new annotations for those of them still waiting to be assigned a function by experimental analysis.

The method was applied to over 14 million protein sequences, including 988 genomes and the UniprotKB release 2011_12.

2.4.1 Applications

Besides the straightforward usage of the method, i.e. the annotation of new protein sequences, it was used for other tasks.

One was relative to the identification of magnesium binding sites in human proteins [52], by using Hidden Markov Model specific to the BAR+ clusters. In this way, the binding locations of magnesium can be predicted for human proteins that are still to be analyzed. Magnesium is a ion involved in several biological processes and then it can be associated to molecular pathogeneses. The system allows the inheritance of annotation not only from other human protein sequences but also from protein sequences belonging to other organisms, also involved in magnesium binding. Another application was for the annotation of Sus Scrofa (pig) proteome [54]. In this case, the dataset of BAR+ was enriched with more sequences specific to the organism of interest. Then, thanks to the statistical validation of clusters, a separate Web service for annotation was developed. Both the human and the pig examples highlight the flexibility of the method. While the human example is quite straightforward in its common interest, since the most studied and crucial diseases and biological functions are those of the human species, the pig case study suggests that also the agricultural field is looking for such tools. This is due mostly because identifying strengths and weaknesses of species intended for

human consumption is crucial for the food industry. An animal or plant with a better yield thanks to specific properties of its genome can improve both the cash flow of the industry and the availability of food for the consumers. On the other end, pinpointing the weaknesses and potential diseases allows to take action against possible food poisoning or epidemics that could diminish the availability of food.

Chapter 3

Community detection

The first possible improvement for BAR+ presented in this work is related to its ability to assign annotations that are very specific, instead of general ones. Since the same annotation may be present in different clusters, the method may be biased towards validating general annotations, i.e. shallow GO terms. A very general term, like "binding" or "transport" is of little use for the scientists who would benefit of the added information conveyed by an automatic annotation system. On the other end, very specific but conflicting annotations may be validated in the same cluster, leading to confusion and little information about the function of a previously not annotated sequence. In the example of GO terms, if the methods validates many terms on the same cluster and those terms are deep in the ontology and share a common ancestor just a few hops away, it is difficult to pinpoint the correct one for the sequence of interest. I.e. if we get a validated annotation for "iron binding", "vitamin binding" and "sulfate binding", it is unlikely that the protein we are interested in is going to bind all of them.

A fine grained annotation method could lead to a better, more accurate and useful annotation. Given that the method relies on a graph of similarity relationships, a way to identify subsets of sequences with higher similarity between them with respect to their similarity to the other sequences in the cluster could be a solution. Since similar problem was studied in social sciences, approaches from that field have been tested.

3.1 Graph theory and terminology

A graph $G(V, E)$ is defined as a set V of n vertices, also called nodes, and a set $E \subseteq V \times V$ of m edges connecting pairs of vertices. Edges may be associated to a weight representing a quantitative quality of the relationship between vertices, like the strength of the connection, the length of the path between the two nodes or something else. Vertices connected by an edge are said to be neighbours.

Unweighted graphs can be thought as a special class of weighted graphs in which edges can have a weight of 0 or 1. A graph is dense if the number of the number of edges is proportional to n^2, otherwise it is sparse. A graph can be directed or undirected: given a pair of vertices, the graph is directed if the order of vertices in the pair matters, i.e. the edge starts from the first one and ends on the second. Otherwise, if the order does not matter, the graph is said to be undirected.

One way to store the edges is by using the adjacency matrix, an n by n matrix whose cell in the i-th row and j-th column contains the weight of the edge from vertex i to vertex j. Obviously, the adjacency matrix of an undirected graph is symmetric.

The degree k of a vertex is the number of edges or the sum of the weights of the edges (strength) connecting it to other vertices. If we denote the adjacency matrix with A and the degree of vertex i with k_i, the degree is given by 3.1.

$$k_i = \sum_j A_{i,j} \tag{3.1}$$

A path is an ordered sequence of edges where each edge starts from the ending vertex of the previous one. A shortest path between two vertices is one so that the sum of the weights of its edges is the minimum among all the weights of the possible paths between those two vertices.

A component in a graph is a set of nodes that can be reached from each other using a path. A graph is partitioned if it is composed by more than one component.

3.2 Communities in graphs

Social networks like Facebook[1] and Twitter[2] allow the analysis of very large graphs representing social interactions of people. One topic of interest, studied well before the advent of such platforms, is the identification of tightly connected groups of people inside larger social graphs. As an example, if the graph represents the work interactions between people, a person would belong to the group of its co-workers and colleagues more than to the group of people she is doing business with. Such groups are called communities and may identify subsets of neighbours, fellow chess players or customers with similar spending habits.

The first and most cited example of such identification is the Zachary's Karate Club [66]. The analysis started from a group of 34 athletes training at the same karate club. The group eventually broke up in two, due to some personal issues between the instructors.

The graph was built setting the members of the club as vertices and adding edges representing a friendship relation between them. Given that the outcome of the split is known, the goodness of different approaches in identifying the two groups can be tested. Figure 3.1 shows a graphical representation of the graph, the two groups are highlighted by different shapes and colors for the vertices.

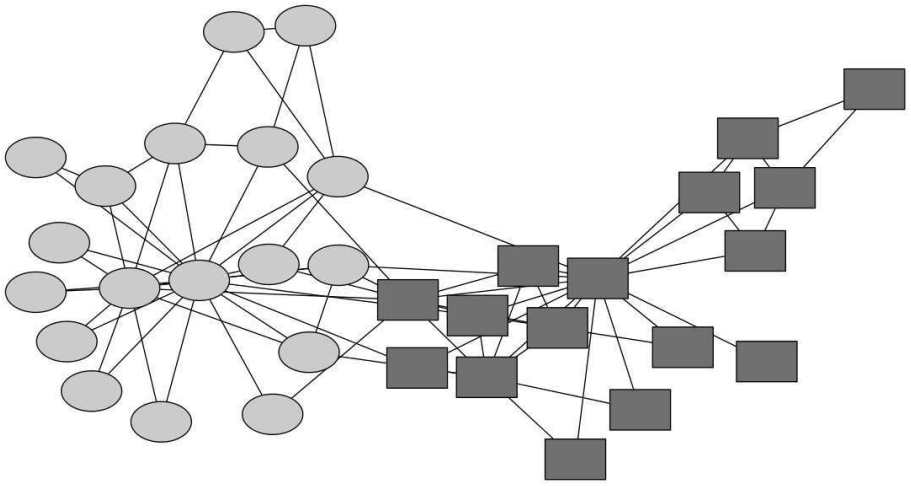

Figure 3.1: A visual representation of Zachary's karate club. Each node is an athlete, different node shapes and colors are assigned to different communities.

[1] www.facebook.com
[2] www.twitter.com

3.3 Modularity

Despite the usefulness of the Zachary's karate club network as testbed, in real world applications the identification of communities is the goal and there is no a priori knowledge of the actual composition of the communities, let alone their number.

Newman, one of the most prolific scientists working on the community identification problem, proposed a mathematical function evaluating how good a specific partitioning of the graph would be. The core idea was that vertices in the same community should have more connections among each other than edges towards members of other communities. Newman called this "modularity function" and formalised it using two similar formulas.

Given that A is the adjacency matrix, so that if vertex i is connected to vertex j then $A_{i,j}$ contains 1 if the graph is unweighted, the weight of the edge if the graph is weighted, or zero if there is no edge between the two. k_i is the degree of vertex i, i.e. the number of edges from vertex i to other vertices. In the case of a weighted graph, k_i is the strength of the node, i.e. the sum of the weights of the edges from vertex i to other vertices (eq. 3.1).

The sum of the degrees of all vertices, or the sum of their strengths in the case of a weighted graph, is $2m$[3]

$$2m = \sum_{i,j} A_{i,j} \tag{3.2}$$

c_i is the community associated to vertex i, while $\delta(a, b)$ returns 1 if a and b have the same value. The modularity measure can then be expressed [21] as in equation 3.3.

$$Q = \frac{1}{2} \sum_{i,j} \left(A_{i,j} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \tag{3.3}$$

The modularity embeds the null model: $\frac{k_i k_j}{2m}$ is the probability that an edge connects vertex i with vertex j. The value of Q ranges from -0.5 to 1, depending on the topology of the graph. Higher values are associated to a better partitioning of the graph, i.e. the assignment to the

[3] notice that for unweighted graphs this value is exactly the double of the number of edges, while for weighted graphs there may be no relation with the actual value of m

communities is better given 3.3.

As an example, if we consider a graph composed by five vertices each connected to all the others (a clique), we get that if we place all the vertices in the same community, the value of the modularity measure Q is equal to zero. Any assignment different from this one leads to a decreased modularity value, due to the fact that the modularity is evaluated as follows.

$$Q = \frac{1}{2}\sum_{i,j}\left(A_{i,j} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j) \tag{3.4}$$

$$= \frac{1}{2}\sum_{i,j}\left(A_{i,j} - \frac{4\cdot 4}{2\cdot 10}\right) \tag{3.5}$$

$$= \frac{1}{2}\sum_{i,j}\left(A_{i,j} - \frac{4}{5}\right) \tag{3.6}$$

$$= \frac{1}{2}\cdot 5\cdot\left[4\cdot\left(1-\frac{4}{5}\right) - \frac{4}{5}\right] \tag{3.7}$$

$$= \frac{1}{2}\cdot 5\cdot\left[\frac{4}{5} - \frac{4}{5}\right] \tag{3.8}$$

$$= 0 \tag{3.9}$$

Where in 3.5 , $\delta(c_i, c_j) = 1$ for every i and j. In 3.7 we have $\left(1-\frac{4}{5}\right)$ for four elements in the row and one $-\frac{4}{5}$ for the missing self loop, all multiplied by 5 since it applies for every row in the matrix.

If we move one node to a new community, the value of Q decreases since we are going to miss the positive contribution of some edges. In the specific case, 3.7 is going to have $3\cdot\left(1-\frac{4}{5}\right)$, leading to a $-\frac{1}{5}$ for each row, and then to $Q = -0.5$.

3.3.1 Limits of modularity maximisation

False communities

Even if conceptually ideal, maximising modularity may not lead to the best partition. This is due to the fact that the modularity measure increases when a node is not left in a separate community. This can be easily seen using an example similar to the previous one. If we build

a graph starting from a clique and then add new vertices, each one connected to a different vertex from the original clique, maximising the modularity may lead to nonsensical results. As an example, we take the five vertices forming a clique and then we add other five as described previously, we obtain a graph like the one depicted in figure 3.2.

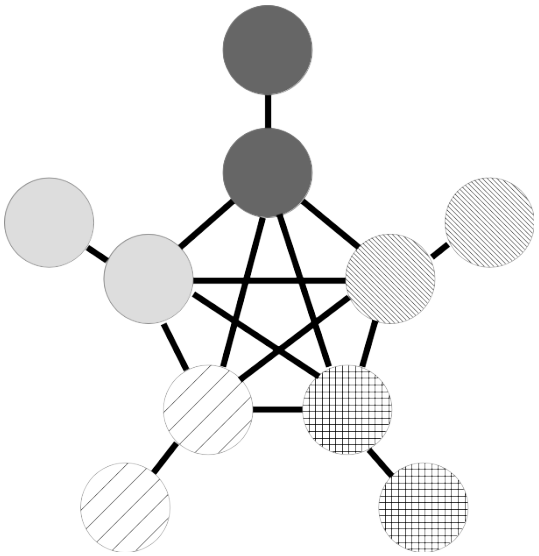

Figure 3.2: False communities due to modularity maximization.

Since the vertices in the clique are all connected to each other, common sense suggests that they form a community. This community may or may not include the new vertices. However, to maximise the modularity measure we need to put each vertex in the clique in a separate community, shared with the new "external" vertex. With this assignment we get that $Q = 0.133$, higher than the $Q = 0$ we could get by putting all the vertices in the same community.

Resolution limit

Another limitation is given by the so called "resolution limit" of modularity maximisation [30]. If we consider a graph composed by a set of cliques arranged in a ring, we can see that the problem presented shows up again, but at a different scale. Instead of having the highest modularity by having separate communities for vertices in different cliques, we obtain communities for pairs of cliques. This is tied to the topology of the graph, if we denote the number of cliques as k and the number of vertices inside each clique as s, the modularity of a

partitioning created by assigning each clique to a different community is bigger than the one with communities including pairs of cliques only if equation 3.10 is satisfied.

$$s(s-1)+2>k \tag{3.10}$$

Complexity

Another big issue with the modularity measure is that maximising it is a NP-Hard problem [15]. We can then exploit algorithms and techniques that approximate the maximum, thanks to heuristics or some desirable property for the resulting partitioning.

3.4 Community detection algorithms

The number of community detection algorithms is quite high, for a complete survey please see [29]. The methods used range from passing properties from one vertex to the other, to stochastic methods and to others that manipulate the adjacency matrix in some way.

At the core of all these methods there is the assumption that the adjacency matrix represents the similarity between two vertices. All the methods considered in this work are based on modularity maximization, for consistency purposes. Other methods based on message passing of spreading of some property have been tested but they are not included in this dissertation.

3.4.1 Girvan-Newman

The Girvan-Newman algorithm [33, 43] starts from a straightforward concept: there should be less edges between vertices in different communities than between edges inside the same community. Using a metric to identify edges connecting different, it would be possible to remove them from the graph, eventually having a set of disconnected components, each one representing a community. In that way, the algorithm is a divisive one: while iterating it subsequently splits

the graph in disconnected components, up to the limit of completely removing all the edges and leaving only isolated vertices.

The metric proposed by Girvan and Newman to select the edge to remove is a centrality one, the betweenness centrality [3, 32]. After computing all pairs shortest path, i.e. all the shortest paths between every couple of vertices, the betweenness measures the ratio of those passing through a specific vertex or edge over the total number. In this way, the higher the betweenness is, the more important the edge is in connecting different communities.

The computational complexity of this method depends on the computational complexity of the betweenness evaluation. Using the faster algorithm to date, developed by Brandes [16], the overall complexity of the Girvan-Newman algorithm is $O(nm^2 + mn^2 \log n)$ for weighted graphs, with n and m as defined in section 3.1.

3.4.2 Louvain method

The Louvain method [9] is an aggregative approach. Similar to the method by Clauset [21] and the one by Newman and Girvan [43] (different from the one presented in 3.4.1), it mimics the construction of the dendogram from the leaves to the root.

The algorithm starts with each vertex assigned to a different community. It then proceeds as follows:

1. evaluate the increase of modularity that would occur by putting adjacent nodes in the same community
2. choose the best pair from step 1 and actually assign the two nodes to the same community
3. consider the new community as a single node
4. go back to step 1.

The procedure ends when it is not possible to further increase the modularity. The exact

computational complexity of the algorithm has not been calculated, but it is roughly estimated to be $O(n \log n)$, with n the number of vertices.

3.5 Libraries

The software tools checked were some of those exposing an Application Programming Interface (API), making easier to build and analyse big graphs like the case study presented in 3.6. The most important features considered were speed and easiness of interaction and extension. There are many graph libraries available, but a comprehensive survey of them was beyond the scope of the research presented here.

It was also taken into account the fact that most computational biologists and bioinformaticians prefer interpreted languages like Python thanks to their simple syntax, fast execution model and to the availability of well known and tested software libraries like BioPython [23] and BioPerl [62]. This was another key element in identifying potential libraries: future modifications and extensions of the system should be possible also to researchers lacking strong programming skills and a computer science background.

3.5.1 WebGraph

The WebGraph Framework [11, 12] is a library designed for managing graphs representing hyperlinked pages in the Web. The main feature of this library is the high compression achieved by its internal representation of the graph. WebGraph uses many strategies to achieve a very small memory footprint for the graph, exploiting properties such as locality, similarity among vertices and representation techniques like distances among vertex identifiers instead of their actual identifiers and copy blocks for replicating similar adjacency lists. Using WebGraph, huge graphs with a structure similar to the World Wide Web can be stored in few kilobytes and traversed fast.

However, graphs in WebGraph are static and cannot be manipulated. For example, while

betweenness centrality can be evaluated, even if only on nodes, the Girvan-Newman algorithm previously described (section 3.4.1) cannot, since it would require the removal of edges.

3.5.2 NetworkX

NetworkX[4] is a graph manipulation library written in Python.

The Louvain method is available as a separate download for NetworkX. The implementation is quite fast since it uses many support data structure to speed up calculations by re-using data. Memory usage for the test cases was quite high compared to other libraries tested, but it wasn't the worst one.

3.5.3 Gephi

Gephi [6] is a graph visualisation and manipulation tool. Written in Java, it is targeted more towards social scientists and people who wants to graphically manipulate their data and produce visually appealing images rather than to perform deep mathematical analysis and batch execution. The developers released also a "toolkit" with API exposing core functionalities for batch execution, however this toolkit has some drawbacks detailed in the following discussion.

The major version tested in this work is numbered 0.8 and contains an implementation of the Louvain method discussed in section 3.4.2. The communities of the test case presented in section 3.6.3 were evaluated using this library. However, some issues have been identified by testing this library.

The first issue was the lack of support for edge weights in the computation of modularity. This feature was finally added in version 0.8.1-alpha thanks to the ability to directly reach the developers via a Web discussion forum.

The second and most important issue is the underlying implementation of the graphs in Gephi. Since it is oriented to a graphical representation and to a direct interaction via a Graphical User

[4] Available at https://networkx.github.io/

Interface (GUI), data structures used to the purpose of drawing the graph on screen occupy a lot of memory. Spatial data structures, like quad-trees, and others rapidly fill the memory when graph size increases. Even on the test server, a machine equipped with 256 Gigabytes of RAM, running analyses on graphs bigger than the case study presented in section 3.6.3 was not possible using Gephi.

3.5.4 graph-tool

graph-tool is a graph library [48] with features similar to NetworkX. It is mostly implemented in C++ using the Boost Graph Library, with many parts of the code exploiting parallel computations thanks to OpenMP [18]. The API is in Python, to simplify the access to the complex data structures behind it, that heavily rely on C++ templates.

Thanks to the fact that the library is open source, it was possible to fix an error in its modularity implementation. Also, the betweenness centrality measure is evaluated using a C++ implementation of the Brandes algorithm [16], making it a very fast one.

3.6 Communities in BAR+ clusters

The basic idea behind this research was that detecting communities inside BAR+ clusters could improve the level of detail of annotations. Isolating specific GO terms into specific communities inside a cluster could be helpful in returning them to the user as more reliable. Besides that, the discovery of smaller subsets of proteins with a stronger relationship with respect to the already strict constraints of the BAR+ clustering method may lead to further insights on their functional, structural and phylogenetic attributes.

3.6.1 Choosing the library

To the purpose of this work, different libraries have been tested. Some of them are presented in section 3.5. At first, given the size of the graphs, WebGraph (3.5.1) was selected for the small footprint of the data. However, the betweenness centrality (3.4.1) implemented in WebGraph was evaluated only on vertices and not on edges. A possible solution was to transform the graph to its line graph, i.e. the graph in which every vertex of the original one becomes an edge and every edge a vertex. Another problem in using the Girvan-Newman algorithm on WebGraph is that the graph must can't be dynamically modified, due to all the compression techniques used in storing it. Iterating the betweenness calculation and edge removal led to a very slow computation, since the graph should be rebuilt from scratch every time.

The actual libraries used and their use cases are described in the following.

3.6.2 First tests

A first set of tests were conducted on small clusters after implementing the the Girvan-Newman algorithm (3.4.1) on graph-tool (3.5.4). One cluster, identified as #4051 and containing 263 sequences showed that the community detection algorithm had some potential in isolating both specific GO terms and EC numbers. The computational complexity of the algorithm, mostly due to the fact that the betweenness centrality needed to be recalculated at every iteration, made it unfeasible for application on bigger clusters.

The faster Louvain method was then selected for a larger scale test. Not only its speed was taken into account, but also its performance in comparative analysis [36]. The implementation of the method in the Gephi library was applied to all clusters with more than 100 protein sequences. The distribution of communities in the clusters is shown in table 3.1: the number of clusters containing a specific number of communities is shown along with that number of communities identified.

Then a statistical validation, using the same Bonferroni-correction for evaluating the p-value

as used in the original BAR+ validation, was performed.

However, manually checking over ninety thousand clusters was unfeasible, let alone the fact that many of them lack the variety of annotation of the bigger ones. Then, the main focus was on the bigger cluster, composed of more than 87893 protein sequences.

3.6.3 Case study: ABC transporters and biological processes

The biggest cluster of BAR+ considered in the preliminary evaluation contains 87893 sequences, mainly from Prokaryotes.

Annotations from Gene Ontology [4], from Pfam [28], and the 22 PDB structure associated to the cluster indicate that the cluster contains sequences of the ATP-binding domain of the ABC transporters.

As reported in [53], the ATP-binding cassette transporters (ABC-transporter) are members of a protein superfamily that is one of the largest and most ancient family with representatives in all organisms from prokaryotes to humans [2, 8]. ABC transporters include transmembrane proteins that couple the Gibbs free energy of Adenosine TriPhosphate (ATP) hydrolysis to translocation of various substrates across membranes. ATP-binding cassette is also responsible for non transport-related processes such as translation of RNA and DNA repair [26, 64]. Their correct annotation in different newly sequenced genomes is therefore relevant to address several issues including also tumor resistance, cystic fibrosis, bacterial multidrug resistance, and a range of other inherited human diseases. The Transporter Classification DataBase [61] is a curated database where transporters are classified in relation to their main function and source organisms. Here, the ABC transporter superfamily comprises 1 073 sequences, 6.2% of which is endowed with a three-dimensional structure in the PDB database. In turn UniProtKB release 2011_12, the major resource of protein sequences freely available, lists 390 628 sequences under the query abc transporter of which about 99% have been only annotated on the basis of sequence similarity. The problem is therefore how to extend the type of annotation from well-known proteins to putative ones.

# communities	**# clusters**
1	8233
2	19491
3	14770
4	5689
5	1939
6	762
7	399
8	224
9	137
10	97
11	55
12	40
13	22
14	7
15	8
16	5
17	9
18	6
19	2
20	2
21	1
22	3
23	1
24	2
25	2
26	2
28	1
30	1
31	1
35	1
36	1
39	1
40	1
45	1
50	1
82	1

Table 3.1: Distribution of communities in BAR+ clusters.

The most populated cluster of BAR (cluster #1) contains 87 893 protein sequences with only 69 sequences from Eukaryotes (average protein length (281 ± 16%) residues). The cluster contains 22 PDB structures from Prokaryotes (the Root Mean Square Deviation (RSMD) of the backbone of all structures is 0.189nm with 0.039nm Standard Deviation (SD)). 56 448 sequences, including only 44 from Eukaryotes (plantae and algae) are endowed with 292 GO terms and 11 Pfam. After statistical validation the systems lists 73 GO terms (55 molecular function; 14 biological processes; and 6 Pfam terms) (P-value < 0.01). The most frequent and validated Pfam term (carried along by the largest number of sequences) is "ABC tran" (PF00005), corresponding to the ATPbinding domain of the ABC transporters.

ABC transporters belong to the ATP-binding cassette superfamily, involved in the export and import of a wide variety of substrates ranging from small ions to macromolecules. Using BAR+ procedure the remaining 31 445 sequences of the cluster inherit by transfer all the validated GO and Pfam terms. These comprise 25 sequences from Eukaryotes not annotated before, including 22 from Xenopus tropicalis, the only animal in the cluster.

The statistically validated GO terms are quite specific but relative to different substrates. It is quite unlikely that the same protein could be involved in transporting 5 or more different ions. Using community detection was then an interesting test to see if the different substrates would be isolated in different communities.

The Louvain method identified 50 communities in the cluster (with a modularity of 0.99), each including from 5 up to 10 333 sequences (see figure 3.4) [56, 53]. Distribution of sequences among the 50 communities is shown in figure 3.5. Differently from the most general Biological Processes GO terms associated to the cluster, some specific biological processes are populating specific communities:

- Sulfate transport (GO:0008272);
- Nitrate transport (GO:0015706);
- Cobalt ion transport (GO:0006824);

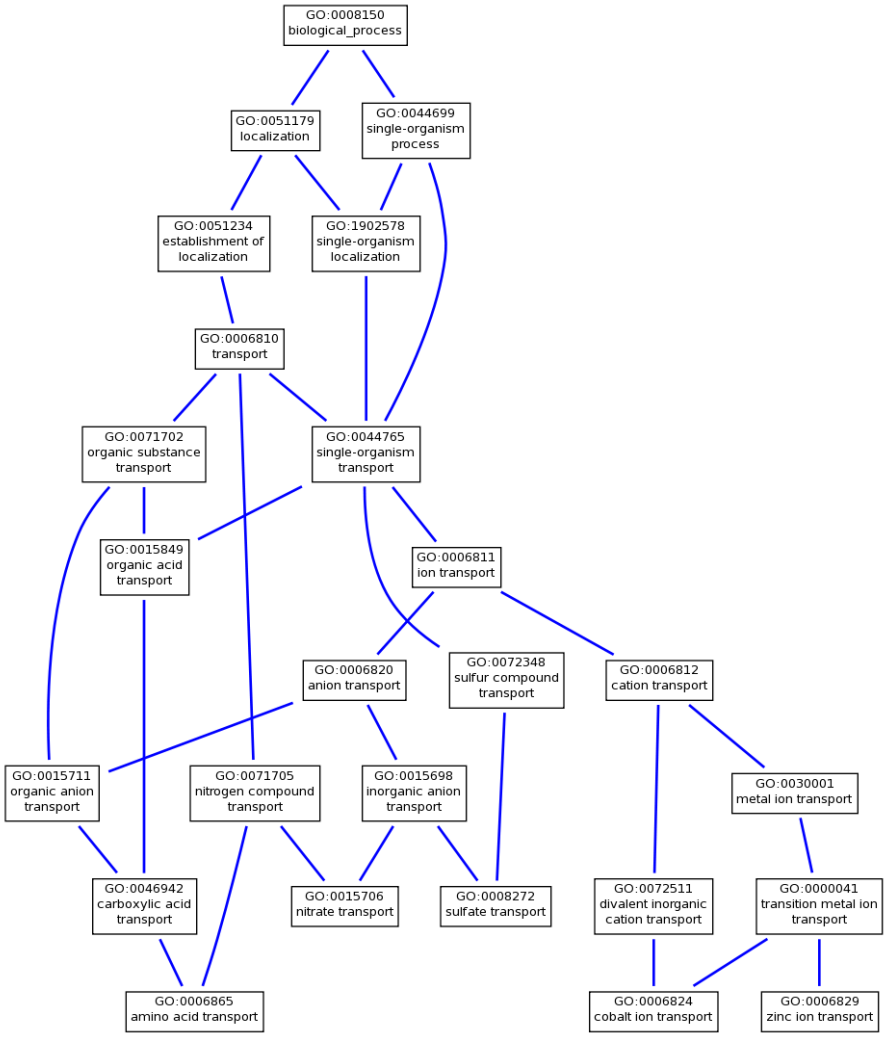

Figure 3.3: View of a some Gene Ontology terms relative to BAR+ cluster #1.

- Zinc ion transport (GO:0006829);
- Amino acid transport (GO:0006865).

As show in figure 3.3, these terms are quite specific but distinct: while some are “siblings”, others belong to a different branch.

In Figure 3.5 some bars are labelled to indicate which community is more associated to a specific transport, and in Table 3.2 the percentage of sequences carrying along the information in each community is also indicated.

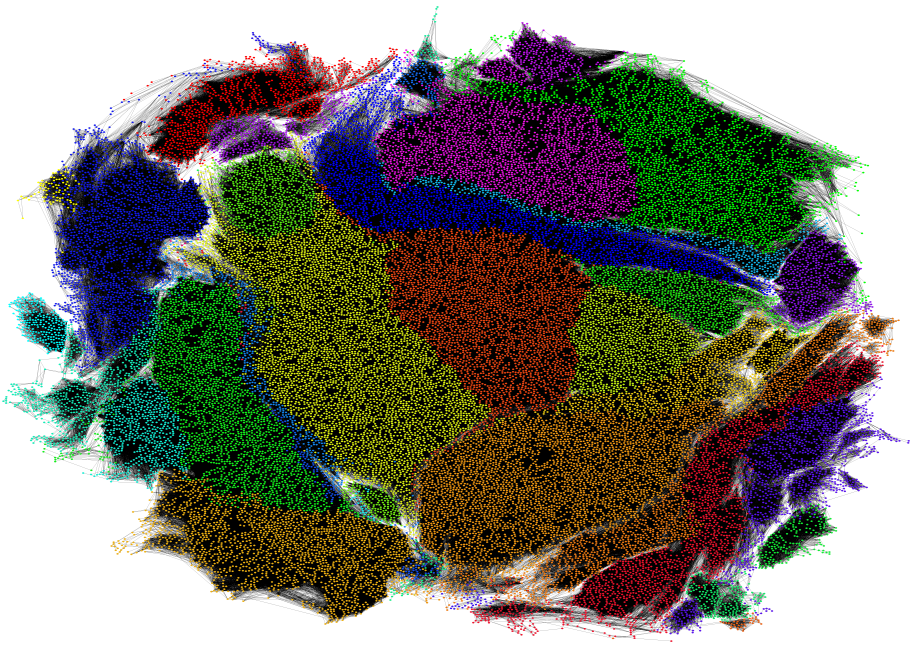

Figure 3.4: Visual representation of the communities in BAR+ cluster #1. Each community is highlighted with a color different from its neighbours.

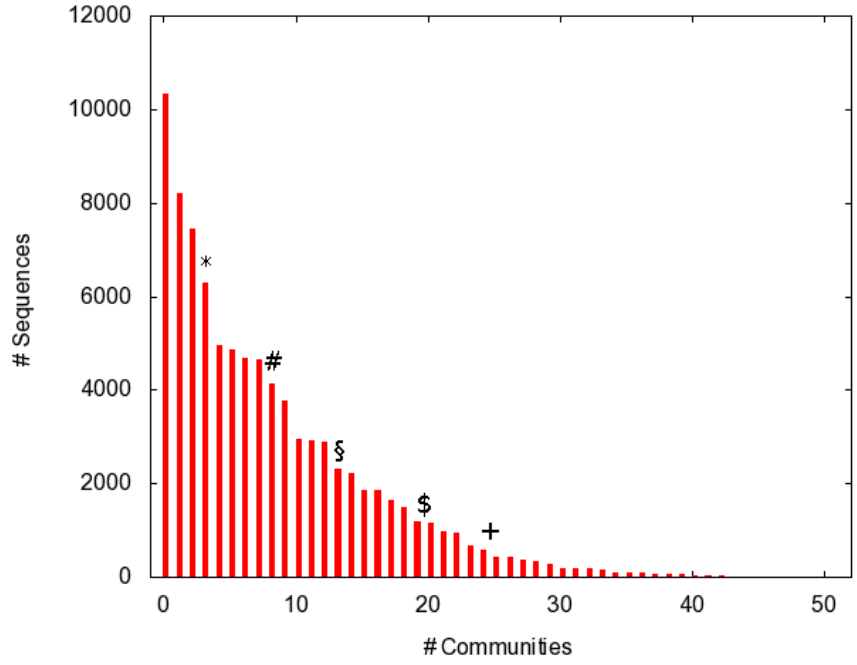

Figure 3.5: Number of sequences per community in BAR+ cluster #1. The most representative biological processes inside the communities are also indicated: (*) Sulfate transport. (#) Nitrate transport. (§) Cobalt ion transport. ($) Amino acid transport. (+) Zinc ion transport.

Transport type	Community	% Seqs
Amino acid	#22	31.6%
Amino acid	#42	54%
Cobalt ion	#15	42.1%
Nitrate	#13	4%
Sulfate	#10	8.4%
Zinc ion	#30	64%

Table 3.2: Distribution of GO terms in communities for the ABC-transporters cluster.

3.6.4 Case study: molecular functions

The same increase in the annotation level can be reached for the Molecular Function terms [58]. As an example the community detection in cluster #9, containing 9 245 sequences mostly from Prokariota, is described in the following. 322 of these protein sequences are annotated in SwissProt [10] for Molecular function and the BAR+ statistically validated annotation includes different transaminase activities.

The community detection identified 17 communities, with sizes ranging from 15 to over 1 700 sequences. 13 communities contain proteins annotated for Molecular Function: after the statistical validation inside each community a most specific transaminase activity can be assigned to each annotated community. Only the transaminase activities validated at cluster level were considered for the analysis at community level. After the statistical validation, more than 5 000 sequences in the communities listed in table 3.3 inherit annotations from the originally annotated 224 SwissProt sequences.

As Table 3.3 shows, some community is marked with more than one validated and specific GO term. This happens because sequences inside the community are annotated in SwissProt with both GO Terms. The algorithm can't separate the two annotations, given the fact that they are both assigned to the same protein sequence.

These are examples of the data analysis possible after partitioning BAR+ clusters into communities. The more specific results, compared to the already validated annotations, are clearly visible in the partitioned clusters analysed.

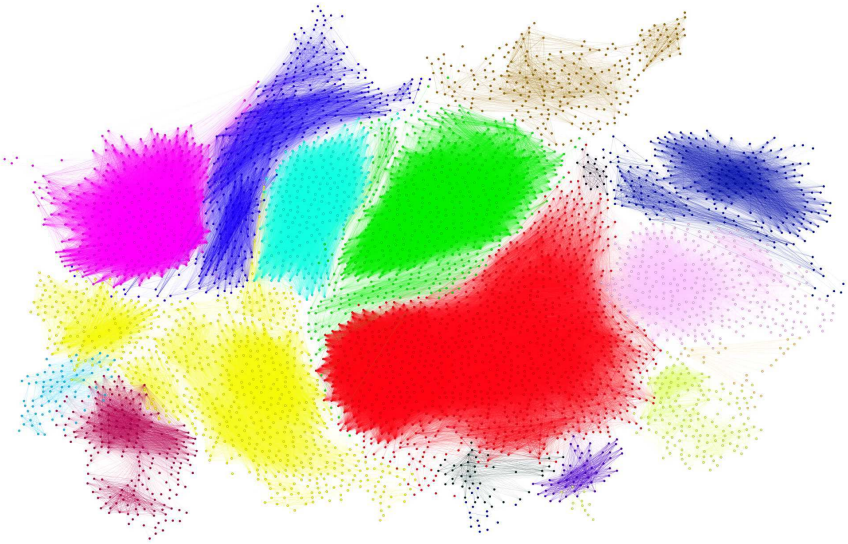

Figure 3.6: Visual representation of the communities in BAR+ cluster #9. Each community is highlighted with a color different from its neighbours.

GO Term	**Description**	**Comm.**
GO:0047305	(R)-3-amino-2-methylpropionate-pyruvate transaminase activity	14
GO:0047298	(S)-3-amino-2-methylpropionate transaminase activity	16
GO:0003867	4-aminobutyrate transaminase activity	16
GO:0004015	adenosylmethionine-8-amino-7-oxononanoate transaminase activity	2,6
GO:0008453	alanine-glyoxylate transaminase activity	14
GO:0033094	butane-1,4-diamine:2-oxoglutarate aminotransferase activity	10
GO:0045303	diaminobutyrate-2-oxoglutarate transaminase activity	5
GO:0047307	diaminobutyrate-pyruvate transaminase activity	5
GO:0003992	N2-acetyl-L-ornithine:2-oxoglutarate 5-aminotransferase activity	15
GO:0004587	ornithine-oxo-acid transaminase activity	13
GO:0009016	succinyldiaminopimelate transaminase activity	15
GO:0043825	succinylornithine transaminase activity	15

Table 3.3: Validation of molecular function GO terms in communities for cluster #9.

Chapter 4

Data generation

At the basis of the previously described annotation procedure, there is an all pair comparison of protein sequences. This comparison allows the creation of the similarity graph and then to the clustering of sequences. The similarity graph is also used to improve the technique using algorithms described later.

Given that the alignment data present in the previous version of the BAR+ database was generated with some difficulties, a brand new all pairs comparison was started to avoid missing links and update the graph to a more recent dataset of sequences. The alignment with the software BLAST [1] was performed using the National Grid Infrastructure (NGI) maintained by the National Centre of INFN (National Institute of Nuclear Physics) for Research and Development into the field of Information Technologies applied to High-Energy physics experiments (CNAF).

It should be noted that the Uniprot consortium distributes a set of already grouped sequences, called UniRef [63]. Each UniRef release groups together sequences with a specific sequence identity score: UniRef 90 for 90% SI, UniRef 50 for 50% and so on. However, these sets are quite different from the ones needed by BAR+ method: in BAR+ there are both a SI constraint and a coverage constraint. While the sequence identity constraint may be mapped to a specific version of UniRef, the coverage can't. Up until 2013, UniRef clusters contained sequences with a variety of lengths, rendering them unusable for BAR+. From 2013, an 80% overlap between

the sequences in the cluster and the longest sequence in the cluster has been implemented, but also this approach is not compatible with the approach of BAR+: besides the difference in the threshold (80% vs 90%), the overlap is evaluated only on the longest sequence, while in BAR+ this threshold is evaluated on each pair of sequences.

The results presented in this chapter have been submitted to a peer reviewed journal, acceptance is pending at the moment of writing.

4.1 All pairs alignment on grid

The distributed grid infrastructure was chosen thanks to its ability to run multiple jobs in parallel, exploiting different computing sites and storing the results into reliable repositories. The grid is mostly used to analyse data from high energy physics experiments, but it is composed by a set of layers, called "tiers", devoted to specific tasks. At the time of usage, Tier 1 and Tier 2 were used to analyse the results of the CERN experiments performed at the Large Hadron Collider, like Compact Muon Solenoid (CMS), that led to confirmation of the existence of the Higgs Boson. The remaining grid sites were available for opportunistic usage without the boundaries of a specific project in the high energy physics domain.

The dataset used to build the new similarity graph was the Uniprot release 2012_12, the last one available at the time. To parallelize the workload, the sequences were split in multiple query sets, each one to be aligned to the whole dataset, called reference database in BLAST terminology. The identified best size for the query subsets was 50 sequences: after testing an average execution time, this was the size best suited to the constraints of grid computing elements. Each job would run more than one subset against the database, sending the results to the storage element. This was possible thanks to a functionality called pilot job: a single parametric job that would spawn a number of children. This was primarily used to decrease the number of data transfers from the storage elements to the computing elements, due to the size of the reference database.

With this system, it was also possible to take care of possible failures: the pilot job would keep

running on a specific subset up to a certain number of retries. The check for a good result was performed by looking at some specific output files. Besides the pilot job checks, each dataset was used by more than one of them. The runs were performed by having a small number of overlaps in the data between different pilot jobs. After the first set of runs, the overlapping portion was increased, to keep filling possible holes in the results.

However, the amount of data and workload proved to be too intensive and demanding for the available grid sites. One problem was the transfer of the reference database: very often the transfer failed, or the local storage was too small to host it. Also, the reliability of the identified sites was low, with major issues with the storage element in Bari. Identifying and trying to solve such issues required a couple of months of testing and debugging.

To overcome these problems, with the help of Daniele Cesini of INFN-CNAF, we started to use a single grid site located in Bologna with an almost unused computing power of 300 cores. Having support on-site was helpful in solving the connection issues. The infrastructure in Bologna had some troubles too, due to a Network File System used by all the machines: the frequent access and transfer of data led to a communication bottleneck. The solution identified was to create a RAM disk to store the reference database, in order to avoid the transfer of the data more than once per system.

The whole alignment took about 800.000 computing hours and led to about 2 TeraBytes of raw data to be processed. Due to the technical difficulties described above, the whole process took about seven months.

4.2 Improvements

Given the doubling in number of sequences each year, keeping this pace would be unfeasible: a computation taking several months of processing would lead to stale data. To be able to update the similarity graph at least every 6 months, a smarter way to parallelize the workload was needed.

The starting point considered was the post processing constraints: given that the edges used in the graph must have a 40% sequence identity over 90% of alignment coverage, aligning a very short sequence with a longer one would be a waste of time (see section 2.4). An optimisation can be obtained then by partitioning the dataset by sequence lengths. The number of partitions depends on both the number of sequences per length and the computational cost of comparing two sequences.

4.2.1 Definitions

Before discussing the mathematics behind the improvements in the raw data generation, some definitions are needed. The basic concepts arise from the assessment of the number of sequences in the database and their respective length. Then, if k is a sequence length, we define the number of sequences of length k as S_k:

$$S_k = \#\text{sequences of length } k \tag{4.1}$$

The size of all the data of length k is defined as $\sigma(k)$ by equation 4.2:

$$\sigma(k) = k \cdot S_k \tag{4.2}$$

In the following text, instead of using the specific value of 90% coverage, we are going to use the variable δ. Then, in our case study $\delta = 0.9$.

The minimum sequence length considered is denoted by $l_{\min}$, while the maximum one by $l_{\max}$.

4.2.2 Constraints

While sequence identity can't be predicted, alignment coverage can be bounded by considering the best case scenario. The bounds for a sequence of length l are sequences of length $\lceil l \cdot \delta \rceil$

and $\lfloor l \div \delta \rfloor$, given that an alignment as shown in figure 2.2 can't have a better coverage for sequences with a length outside that range.

The computational cost of creating the similarity graph is then bounded by the all-vs-all comparison and the within-subset one. Defining $c_1(u, v)$ the cost function for comparing one sequence of length u with one sequence of length v and with S_w the number of sequences of length w as defined by equation 4.1, we obtain the cost of comparing a sequence of length u with all the sequences of length v:

$$S_v \cdot c_1(u, v) \tag{4.3}$$

Then, comparing one sequence of length u to all the sequences of any length:

$$\sum_{j=l_{\min}}^{l_{\max}} S_j \cdot c_1(u, j) \tag{4.4}$$

Comparing all the sequences of length u to all the sequences of any length is then:

$$S_u \cdot \sum_{j=l_{\min}}^{l_{\max}} S_j \cdot c_1(u, j) \tag{4.5}$$

The cost of comparing all the possible pairs of sequences is then given by equation 4.6:

$$c_{all} = \sum_{i=l_{\min}}^{l_{\max}} S_i \cdot \sum_{j=l_{\min}}^{l_{\max}} S_j \cdot c_1(i, j) \tag{4.6}$$

If we instead consider just the interval bounds defined at the beginning of section 4.2.2, we could restrict the second summation to the interval $[\lceil i \cdot \delta \rceil, \lfloor i \div \delta \rfloor]$, obtaining equation 4.7:

$$c_{\text{subsets}} = \sum_{i=l_{\min}}^{l_{\max}} S_i \cdot \sum_{j=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} S_j \cdot c_1(i, j) \tag{4.7}$$

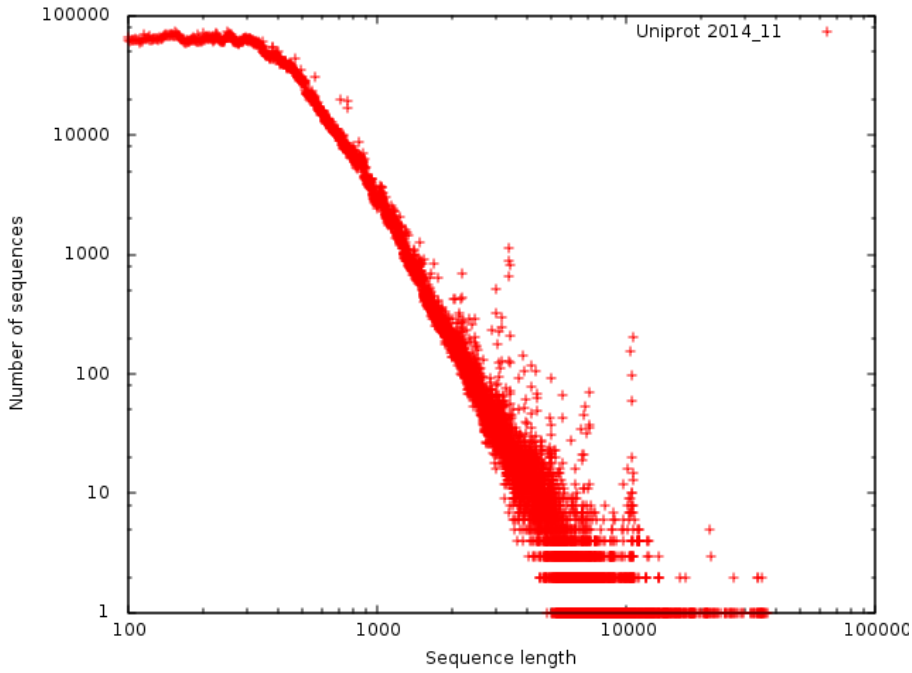

Figure 4.1: Distribution of sequences per length in Uniprot 2014_11. Only lengths greater or equal to 100 are considered. Scale is logarithmic for both axes.

The right side of equation 4.7 is not guaranteed to be the minimum. This is due to the fact that the subsets overlap, leading to duplicated computations. Then we need a way to minimise such duplicated effort, or the constraints we put would be useless. One first straightforward approach was a greedy one: given the distribution of sequences by length, showed in figure 4.1, we take the length l corresponding to the maximum number of sequences. The range $[\lceil l \cdot \delta \rceil, \lfloor l \div \delta \rfloor]$ will be our first subset. We then proceed by selecting the surrounding subsets, i.e. the ones ending at $l-1$ and starting at $l+1$. This approach led, using the data of Uniprot 2014_09, to 14 subsets.

A constraint programming solution was explored, but the tool tested for that purpose (MiniZinc [42]) was not able to manage the space complexity of the problem. Also, tested solutions had a combinatorial complexity, then a different type of approach was devised.

4.2.3 Cost of sequence alignment

The cost function is a core element in identifying the best partition for the data. As defined in section 4.2.1, S_i the number of sequences of length i, $\sigma(i) = i \cdot S_i$ the amount of data (characters) for sequences of length i.

BLAST

Since the software used to perform the sequence alignment is BLAST, we checked its computational complexity and applied it to our scenario. Computational complexity of BLAST is $aW + bN + \frac{NW}{20^W}$, as reported by [1], where W is proportional to the length of the query and N is the amount of data in the reference database. Mapping this to our problem, the computational complexity of aligning a sequence of length k to sequences of length s is:

$$k + \sigma(s) + \frac{\sigma(s) \cdot k}{20^k} \tag{4.8}$$

Given that k should be at least 200, since shorter sequences are less interesting for bioinformaticians, and that the length goes up to over 30 000, the last part of equation 4.8 can be discarded. We then get the following formula, which is equivalent to the one present in equation 4.3:

$$S_s \cdot c_1(k, s) = k + \sigma(s) \tag{4.9}$$

This cost should be applied to all the sequences of length k, and reference database can be generalised to all sequences from length s to length t, obtaining:

$$S_k \cdot \left(k + \sum_{j=s}^{t} \sigma(j) \right) \tag{4.10}$$

If we expand the cost to a whole subset of sequences with lengths ranging from s to t we get

equation 4.11:

$$c(s,t) = \sum_{k=s}^{t} S_k \cdot \left(k + \sum_{j=s}^{t} \sigma(j) \right) \tag{4.11}$$

Equations 4.6 and 4.7 can then be rewritten as follows:

$$\begin{aligned} c(l_{\min}, l_{\max}) &= \sum_{k=l_{\min}}^{l_{\max}} S_k \cdot \left(k + \sum_{j=l_{\min}}^{l_{\max}} \sigma(j) \right) \\ &= \sum_{k=l_{\min}}^{l_{\max}} S_k \cdot \sum_{j=l_{\min}}^{l_{\max}} \sigma(j) + \sum_{k=l_{\min}}^{l_{\max}} S_k \cdot k \\ &= \sum_{k=l_{\min}}^{l_{\max}} S_k \cdot \sum_{j=l_{\min}}^{l_{\max}} \sigma(j) + \sum_{k=l_{\min}}^{l_{\max}} \sigma(k) \\ &= \sum_{k=l_{\min}}^{l_{\max}} \sigma(k) \cdot \left(1 + \sum_{j=l_{\min}}^{l_{\max}} S_j \right) \end{aligned} \tag{4.12}$$

$$\begin{aligned} c(\text{subsets}) &= \sum_{i=l_{\min}}^{l_{\max}} \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} S_k \cdot \left(k + \sum_{j=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} \sigma(j) \right) \\ &= \sum_{i=l_{\min}}^{l_{\max}} \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} S_k \cdot k + \sum_{i=l_{\min}}^{l_{\max}} \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} S_k \cdot \sum_{j=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} \sigma(j) \\ &= \sum_{i=l_{\min}}^{l_{\max}} \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} \sigma(i) + \sum_{i=l_{\min}}^{l_{\max}} \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} S_k \cdot \sum_{j=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} \sigma(j) \\ &= \sum_{i=l_{\min}}^{l_{\max}} \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} \sigma(i) \cdot \left(1 + \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor i \div \delta \rfloor} S_k \right) \end{aligned} \tag{4.13}$$

We would like to find the best partitioning, i.e. the one with the smallest cost for useless comparisons and for duplicated ones due to overlaps between partitions.

4.2.4 Dynamic programming solution

Since the problem of finding the best partitioning is exponential if approached with standard solutions, one possible alternative method may be dynamic programming.

Dynamic programming is widely used to solve very complex problems, from the sequence alignment itself to the knapsack problem. The underlying idea of dynamic programming is to evaluate subsequent steps by starting from previous good solutions. In the sequence alignment example, the Smith-Waterman algorithm builds a matrix of alignment costs, starting from the first position of one sequence and it keeps expanding the solution while scoring it. At the end of the algorithm, tracing back the best scores leads to the best solutions, or more than one in case of ties.

In our case, the cell in the i-th row and j-th column of the matrix stores the cost of aligning sequences from length $\lceil i \cdot \delta \rceil$ to $\lfloor j \div \delta \rfloor$. The idea is to start from the upper right cell, containing the cost for the smallest length and its surroundings given the δ constraint, and see if it would be better to expand the length range in that partition or to start a new range. Identifying a set of cells covering the whole range of lengths and generating the smallest sum would lead to the solution of the problem.

To do that, it is necessary to evaluate the cost of expanding a range so that it would include sequences of higher length. Even more, we want cost to be a function of previously evaluated costs, i.e. costs for cells in the previous column or row. Starting from the cost function 4.11, given $a \leq b \leq c$ we can evaluate such increase as:

$$\mathrm{c}(a,c) = \sum_{k=a}^{c} S_k \left(k + \sum_{i=a}^{c} \sigma(i) \right) \tag{4.14}$$

$$= \sum_{k=a}^{b-1} S_k \left(k + \sum_{i=a}^{c} \sigma(i) \right) + \sum_{k=b}^{c} S_k \left(k + \sum_{i=a}^{c} \sigma(i) \right) \tag{4.15}$$

$$= \sum_{k=a}^{b-1} S_k \left(k + \sum_{i=a}^{b-1} \sigma(i) + \sum_{i=b}^{c} \sigma(i) \right) + \sum_{k=b}^{c} S_k \left(k + \sum_{i=a}^{b-1} \sigma(i) + \sum_{i=b}^{c} \sigma(i) \right) \tag{4.16}$$

$$= \underbrace{\sum_{k=a}^{b-1} S_k \left(k + \sum_{i=a}^{b-1} \sigma(i) \right)}_{\mathrm{c}(a,b-1)} + \sum_{k=a}^{b-1} S_k \sum_{i=b}^{c} \sigma(i) + \underbrace{\sum_{k=b}^{c} S_k \left(k + \sum_{i=b}^{c} \sigma(i) \right)}_{\mathrm{c}(b,c)} + \sum_{k=b}^{c} S_k \sum_{i=a}^{b-1} \sigma(i) \tag{4.17}$$

$$= \mathrm{c}(a, b-1) + \mathrm{c}(b, c) + \sum_{k=a}^{b-1} S_k \sum_{i=b}^{c} \sigma(i) + \sum_{k=b}^{c} S_k \sum_{i=a}^{b-1} \sigma(i) \tag{4.18}$$

Put in words, the cost for the extended partition is the sum of the cost for the previous range, the cost of aligning the sequences for the added range against themselves, and the cost of aligning old sequences versus the new ones.

4.2.5 Algorithm and notes

To speed up the computation, most of the data should be pre-computed and stored in an appropriate data structure. Both S and σ are straightforward, they should be put into arrays indexed by sequence length. However, many elements in the previously shown formulae use summations of contiguous items from those arrays. Then, instead of performing $O(n)$ sums every time, the partial sums could be stored into another array.

With this approach, it is possible to define an array containing in its i-th position the sum of all element of S up to position i. The same applies for sigma and its sum counterpart. For the remainder of this text we introduce the following notations for these two variables:

$$\Sigma_S(t) = \sum_{i=0}^{t} S_i \tag{4.19}$$

$$\Sigma_\sigma(t) = \sum_{i=0}^{t} \sigma(i) \tag{4.20}$$

Expressing the cost function (equation 4.11) in terms of these variables is easy:

$$\mathrm{c}(s,t) = \sum_{k=s}^{t} S_k \cdot \left(k + \sum_{j=s}^{t} \sigma(j) \right)$$

$$
\begin{aligned}
&= \sum_{k=s}^{t} k \cdot S_k + \sum_{k=s}^{t} S_k \cdot \sum_{j=s}^{t} \sigma(j) \\
&= \sum_{k=s}^{t} \sigma(k) + \sum_{k=s}^{t} S_k \cdot \sum_{j=s}^{t} \sigma(j) \\
&= [\Sigma_\sigma(t) - \Sigma_\sigma(s-1)] + [\Sigma_S(t) - \Sigma_S(s-1)] \cdot [\Sigma_\sigma(t) - \Sigma_\sigma(s-1)] \\
&= [\Sigma_\sigma(t) - \Sigma_\sigma(s-1)] \cdot [1 + \Sigma_S(t) - \Sigma_S(s-1)] \qquad (4.21)
\end{aligned}
$$

Costs should be stored too, using a matrix as it is customary in dynamic programming. Since this matrix is an upper triangular one, it is possible to store it into an array of length $n(n+1)/2$. Let's call this matrix M, element $M_{i,j}$ contains $\mathrm{c}(\lceil i \cdot \delta \rceil, \lfloor i \div \delta \rfloor)$. Starting from the main diagonal, it is possible to construct all the elements of a row by using equation 4.21, thanks to the variables previously defined. The diagonal elements are

$$
M_{i,i} = (\Sigma_\sigma(\lfloor i \div \delta \rfloor) - \Sigma_\sigma(\lceil i \cdot \delta \rceil - 1)) \cdot (1 + \Sigma_S(\lfloor i \div \delta \rfloor) - \Sigma_S(\lceil i \cdot \delta \rceil - 1)) \qquad (4.22)
$$

Elements outside the main diagonal can be expressed using formula 4.18 and then applying the result from formula 4.21:

$$
\begin{aligned}
M_{i,j} &= \mathrm{c}(\lceil i \cdot \delta \rceil, \lfloor j \div \delta \rfloor) \\
&= \mathrm{c}(\lceil i \cdot \delta \rceil, \lfloor (j-1) \div \delta \rfloor) + \mathrm{c}(\lfloor (j-1) \div \delta \rfloor + 1, \lfloor j \div \delta \rfloor) + \\
&+ \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor (j-1) \div \delta \rfloor} S_k \sum_{q=\lfloor (j-1) \div \delta \rfloor + 1}^{\lfloor j \div \delta \rfloor} \sigma(q) + \sum_{k=\lfloor (j-1) \div \delta \rfloor + 1}^{\lfloor j \div \delta \rfloor} S_k \sum_{q=\lceil i \cdot \delta \rceil}^{\lfloor (j-1) \div \delta \rfloor} \sigma(q) \\
&= M_{i,j-1} + [\Sigma_\sigma(\lfloor j \div \delta \rfloor) - \Sigma_\sigma(\lfloor (j-1) \div \delta \rfloor)] \cdot [1 + \Sigma_S(\lfloor j \div \delta \rfloor) - \Sigma_S(\lfloor (j-1) \div \delta \rfloor)] \\
&+ [\Sigma_S(\lfloor (j-1) \div \delta \rfloor) - \Sigma_S(\lceil i \cdot \delta \rceil - 1)] \cdot [\Sigma_\sigma(\lfloor j \div \delta \rfloor) - \Sigma_\sigma(\lfloor (j-1) \div \delta \rfloor)] \\
&+ [\Sigma_S(\lfloor j \div \delta \rfloor) - \Sigma_S(\lfloor (j-1) \div \delta \rfloor + 1)] \cdot [\Sigma_\sigma(\lfloor (j-1) \div \delta \rfloor) - \Sigma_\sigma(\lceil i \cdot \delta \rceil - 1)] \\
&= M_{i,j-1} + [1 + \Sigma_S(\lfloor j \div \delta \rfloor) - \Sigma_S(\lceil i \cdot \delta \rceil - 1)] \cdot [\Sigma_\sigma(\lfloor j \div \delta \rfloor) - \Sigma_\sigma(\lfloor (j-1) \div \delta \rfloor)] \\
&+ [\Sigma_S(\lfloor j \div \delta \rfloor) - \Sigma_S(\lfloor (j-1) \div \delta \rfloor + 1)] \cdot [\Sigma_\sigma(\lfloor (j-1) \div \delta \rfloor) - \Sigma_\sigma(\lceil i \cdot \delta \rceil - 1)] \qquad (4.23)
\end{aligned}
$$

However, using formula 4.23 seems overkill with respect to applying formula 4.21 to the definition of $M_{i,j}$:

$$\begin{aligned} M_{i,j} &= \mathrm{c}(\lceil i \cdot \delta \rceil, \lfloor j \div \delta \rfloor) \\ &= [\Sigma_\sigma(\lfloor j \div \delta \rfloor) - \Sigma_\sigma(\lceil i - 1 \cdot \delta \rceil)] \cdot [1 + \Sigma_S(\lfloor j \div \delta \rfloor) - \Sigma_S(\lceil i - 1 \cdot \delta \rceil)] \end{aligned} \tag{4.24}$$

mostly due to the need of looking up for only 4 values and performing three summations and one multiplication, while formula 4.23 makes more accesses to the memory and it performs more computations.

Now, thanks to all the support data structures and pre-computed values, we can focus on the actual dynamic programming solution.

The algorithm tries to expand a solution found in its previous iteration, comparing it to new ones that are locally good. After identifying a good partitioning for the interval $[i, j-1]$, the algorithm checks if the right side of the range should be expanded.

Solutions are represented as a combination of ranges in the length domain. While doing so, it keeps track of the identified ranges. The pseudocode is showed in algorithm 1.

Algorithm 1 Dynamic programming partitioning

```
for i ← 1, N do
    B[i] ← M[1][i]                          ▷ Every range as a local best
    P[i] ← −1                               ▷ Traceback for solutions
    for k ← 2, i do
        tmp ← M[⌊k ÷ δ⌋][i]                 ▷ cost defined as in equation 4.25
        if B[i] > tmp + B[k − 1] then       ▷ new best cost
            B[i] ← tmp + B[k − 1]
            P[i] ← k − 1
        end if
    end for
end for
MinCost ← B[N]                              ▷ The minimum cost found
intervals ← [(P[N], N)]                     ▷ Building the list of intervals
i ← P[N]
while i > 0 do
    intervals.append( (P[i] + 1, ⌊i ÷ δ⌋) )
    i ← P[i]
end while
```

4.2.6 Optimality

It can be proven that algorithm 1 finds the best possible partition for the provided dataset.

The definitions are as follows: the cost function is the same as in equation 4.11, the matrix element $M_{i,j}$ is defined as in the previous section:

$$M_{i,j} = \mathrm{c}\left(\lceil i \cdot \delta \rceil, \lfloor j \div \delta \rfloor\right) \tag{4.25}$$

We then introduce a partitioning up to length k as:

$$P_k = \{[i,j] | [i,j] \text{is a subset in the solution} \wedge j \leq k\} \tag{4.26}$$

and the total cost up to point i as

$$B[i] = \sum_{[s,t] \in P_i} M_{s,t} \tag{4.27}$$

Or, using the approach from the algorithm, we can define $B[i]$ as:

$$B[i] = \min \begin{cases} M_{1,i} \\ \min_{k \leq i} B[k-1] + M_{\lfloor k \div \delta \rfloor, i} \end{cases} \tag{4.28}$$

First, it can be proven that each element in a row of matrix M is greater or equal to than the ones to its left, since:

$$\begin{aligned} M_{i,j+t} &= \mathrm{c}\left(\lceil i \cdot \delta \rceil, \lfloor (j+t) \div \delta \rfloor\right) \\ &= \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor (j+t) \div \delta \rfloor} S_k \cdot \left(k + \sum_{l=\lceil i \cdot \delta \rceil}^{\lfloor (j+t) \div \delta \rfloor} \sigma(l) \right) \\ &= \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} S_k \cdot \left(k + \sum_{l=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} \sigma(l) \right) + \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} S_k \sum_{l=\lfloor j \div \delta \rfloor + 1}^{\lfloor (j+t) \div \delta \rfloor} \sigma(l) + \sum_{k=\lfloor j \div \delta \rfloor + 1}^{\lfloor (j+t) \div \delta \rfloor} S_k \cdot \left(k + \sum_{l=\lceil i \cdot \delta \rceil}^{\lfloor (j+t) \div \delta \rfloor} \sigma(l) \right) \end{aligned}$$

$$
\begin{aligned}
&= \mathrm{c}\left(\lceil i \cdot \delta \rceil, \lfloor j \div \delta \rfloor\right) + \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} S_k \sum_{l=\lfloor j \div \delta \rfloor + 1}^{\lfloor (j+t) \div \delta \rfloor} \sigma(l) + \sum_{k=\lfloor j \div \delta \rfloor + 1}^{\lfloor (j+t) \div \delta \rfloor} S_k \cdot \left(k + \sum_{l=\lceil i \cdot \delta \rceil}^{\lfloor (j+t) \div \delta \rfloor} \sigma(l) \right) \\
&= M_{i,j} + \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} S_k \sum_{l=\lfloor j \div \delta \rfloor + 1}^{\lfloor (j+t) \div \delta \rfloor} \sigma(l) + \sum_{k=\lfloor j \div \delta \rfloor + 1}^{\lfloor (j+t) \div \delta \rfloor} S_k \cdot \left(k + \sum_{l=\lceil i \cdot \delta \rceil}^{\lfloor (j+t) \div \delta \rfloor} \sigma(l) \right) \\
&\geq M_{i,j}
\end{aligned} \tag{4.29}
$$

Elements in the same column follow a similar pattern, with elements near the diagonal having a smaller value than those in the rows above them:

$$
\begin{aligned}
M_{i-t,j} &= \mathrm{c}\left(\lceil (i-t) \cdot \delta \rceil, \lfloor j \div \delta \rfloor\right) \\
&= \sum_{k=\lceil (i-t) \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} S_k \cdot \left(k + \sum_{l=\lceil (i-t) \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} \sigma(l) \right) \\
&= \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} S_k \cdot \left(k + \sum_{l=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} \sigma(l) \right) + \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} S_k \sum_{l=\lceil (i-t) \cdot \delta \rceil}^{\lceil i \cdot \delta \rceil - 1} \sigma(l) + \sum_{k=\lceil (i-t) \cdot \delta \rceil}^{\lceil i \cdot \delta \rceil - 1} S_k \cdot \left(k + \sum_{l=\lceil (i-t) \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} \sigma(l) \right) \\
&= \mathrm{c}\left(\lceil i \cdot \delta \rceil, \lfloor j \div \delta \rfloor\right) + \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} S_k \sum_{l=\lceil (i-t) \cdot \delta \rceil}^{\lceil i \cdot \delta \rceil - 1} \sigma(l) + \sum_{k=\lceil (i-t) \cdot \delta \rceil}^{\lceil i \cdot \delta \rceil - 1} S_k \cdot \left(k + \sum_{l=\lceil (i-t) \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} \sigma(l) \right) \\
&= M_{i,j} + \sum_{k=\lceil i \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} S_k \sum_{l=\lceil (i-t) \cdot \delta \rceil}^{\lceil i \cdot \delta \rceil - 1} \sigma(l) + \sum_{k=\lceil (i-t) \cdot \delta \rceil}^{\lceil i \cdot \delta \rceil - 1} S_k \cdot \left(k + \sum_{l=\lceil (i-t) \cdot \delta \rceil}^{\lfloor j \div \delta \rfloor} \sigma(l) \right) \\
&\geq M_{i,j}
\end{aligned} \tag{4.30}
$$

Now, if we consider B, we can derive the following:

$$
B[j] \geq B[i] \quad \forall j > i \tag{4.31}
$$

The proof is by induction on i. The hypotheses are:

$$
B[1] = M_{1,1}
$$

and

$$B[2] = \min \begin{cases} M_{1,2} \geq M_{1,1} = B[1] \\ B[1] + M_{\lfloor 2 \div \delta \rfloor, 2} \geq B[1] \end{cases}$$

Then, assuming $B[i-1] \leq B[i]$ true, let's prove $B[i] \leq B[i+1]$. By equation 4.28, we have that:

$$B[i+1] = \min \begin{cases} M_{1,i+1} \\ \min_{k \leq i+1} B[k-1] + M_{\lfloor k \div \delta \rfloor, i+1} \end{cases}$$

Since $M_{1,i+1} \geq M_{1,i}$ by proof 4.29, it is greater or equal than $B[i]$ since it has value $M_{1,i}$ or something smaller, given the choice of the minimal result of the two cases. For the second case we use *reduction ab absurdum*. Suppose that exists a $k^* \leq i+1$ so that $B[i+1] < B[i]$:

$$\begin{aligned} B[k^*-1] + M_{\lfloor k^* \div \delta \rfloor, i+1} &< B[n] \\ B[k^*-1] + M_{\lfloor k^* \div \delta \rfloor, i+1} &< B[k^*-1] + M_{\lfloor k^* \div \delta \rfloor, i} \qquad \text{by definition of B[i], 4.28} \\ M_{\lfloor k^* \div \delta \rfloor, i+1} &< M_{\lfloor k^* \div \delta \rfloor, i} \end{aligned} \tag{4.32}$$

But equation 4.32 can't be true, as proven in 4.29.

The last thing to be proven is that the resulting partitioning is the best solution, i.e. $B[l_{max}]$ is minimum.

This can also be proven by induction, simply considering that each $B[i]$ chooses the minimum combination of a previous $B[k]$ with $k < i$, guaranteed to be minimum for index k, and $M_{\lfloor k \div \delta \rfloor, i}$. The value of this second element can't be decreased further, due to the following constraints. First, if one would modify the upper bound, it should be decreased (to exploit the result in equation 4.29), however i generates the upper bound in the cost function, $\lfloor i \div \delta \rfloor$, needed to compare the element i to all its successors up to the constraint: decreasing it could lead to

missing comparisons.

Second, increasing the value of the lower bound $\lfloor k \div \delta \rfloor$, to exploit the result in equation 4.30, would change the overlap between the range specified by $B[k]$ and the one of $B[i]$. With the implemented solution, the overlap is the interval $[k, \lfloor k \div \delta \rfloor]$: since it can't be shrinked by decreasing the upper bound for $B[k]$, by the reasoning just described, the lower bound for M should be increased. With this approach, the overlapping interval would be $[k + \epsilon, \lfloor k \div \delta \rfloor]$, leading to missing comparisons for elements in the range $(k, k + \epsilon)$ with elements in the range $(\lfloor k \div \delta \rfloor, \lfloor (k + \epsilon) \div \delta \rfloor)$.

The last proof is the choice of $k - 1$ as index for B. Given $M_{\lfloor k \div \delta \rfloor, i}$, we want to choose an index $k - t$ so that $\lfloor (k-1) \div \delta \rfloor = \lfloor (k-t) \div \delta \rfloor$. In this way we minimize the overlap of the range extension for the previous solution with the current choice. For the remainder of this demonstration, we define $\delta = \frac{p}{q}$ with $p \leq q$.

$$
\begin{aligned}
\lfloor (k-1) \div \delta \rfloor &= \lfloor (k-t) \div \delta \rfloor \\
\left\lfloor \frac{(k-t)q}{p} \right\rfloor &= \left\lfloor \frac{(k-1)q}{p} \right\rfloor \\
\left\lfloor \frac{ap + c - (t+1)q}{p} \right\rfloor &= \left\lfloor \frac{ap + c}{p} \right\rfloor && \text{with } (k-1)q = ap + c \text{ and } c < p \\
a + \left\lfloor \frac{c - (t+1)q}{p} \right\rfloor &= \left\lfloor \frac{c}{p} \right\rfloor + a \\
\left\lfloor \frac{c - (t+1)q}{p} \right\rfloor &= \left\lfloor \frac{c}{p} \right\rfloor \\
\left\lfloor \frac{c - (t+1)q}{p} \right\rfloor &= 0 && (\text{since } c < p) \\
c &= (t+1)q \\
t + 1 &= \frac{c}{p} \\
t &= \frac{c}{p} - 1 \\
t &= -1 && \text{since } t \in \mathbb{N}
\end{aligned}
$$

Then the upper bound for the previous range is $\lfloor (k-1) \div \delta \rfloor$, leading to an index for B equal to $k-1$. Given that, $B[k-1]$ is the smallest previous solution that overlaps with the current range $M_{\lfloor k \div \delta \rfloor, i}$.

4.2.7 Implementation

The implementation was done using the C++ language. The steps presented in algorithm 1 have been implemented using some strategies to manage both the data and the execution.

First, since the values for σ and the summations are potentially huge, the C data type `uint64_t` may be able to store them, but even if the implementation allowed $2^{64}-1$ as maximum value the actual data is unknown and then value may be higher. Also, allocating memory using directly this data type may be a waste of space for the elements with a smaller value.

To work around these representation problems, the Gnu Multiple Precision Arithmetic Library (GMP)[1] was used. This library defines data structures that can store arbitrary precision numbers, with a variable memory footprint depending on the actual value of the number stored.

As stated in the previous section, the evaluation of the matrix M can be performed in parallel, since every row is independent from the others. The parallel implementation is useful, given that the matrix is in the order of more than 500 000 000 entries, considering a maximum sequence length of about 30 000. Since the algorithm was implemented in the C++ language, the parallel execution is achieved thanks to the OpenMP library[2][18].

The final implementation however does not explicitly compute the whole matrix, instead it evaluates one column at the time. The parallelization in this case can be achieved on the inner for loop, at line 5 in algorithm 1, but since it involves finding a minimum value, the speed up would not be so significant.

[1] Available at https://gmplib.org/ version 6.0.0
[2] Available at http://openmp.org

4.3 Results

4.3.1 Simple test case

An initial test was performed on a toy problem. The dataset was composed by three hundred and one consecutive lengths ranging from 300 to 600, each assigned with one hundred thousand elements.

In this scenario, an all against all approach would lead to a cost of over 4.07×10^{17}, evaluated using the equation 4.12.

The naive all subsets solution, as reported in equation 4.13, has a total cost of 1.1×10^{19}. It could be lowered thanks to the fact that the subsets at the beginning and the end may be merged.

The proposed algorithm identified 6 subsets for a total cost of 2.13×10^{17}, evaluated using 4.11.

There is then an improvement of half of the cost. This improvement is a constant rate as shown in table 4.1, that shows the cost of full range and subsets while varying the number of sequences per length.

#Seqs	All vs All	Naive	# subsets	Best
100000	4.07×10^{17}	1.11×10^{19}	6	2.13×10^{17}
10000	4.07×10^{15}	1.11×10^{17}	6	2.13×10^{15}
1000	4.07×10^{13}	1.11×10^{15}	6	2.13×10^{13}
100	4.07×10^{11}	1.11×10^{13}	6	2.13×10^{11}

Table 4.1: Comparison of alignment costs given different partitioning for a toy dataset. The number of sequences is per length

4.3.2 Actual data test case

The algorithm was executed on the data relative to the Uniprot release 2014_11. From the dataset, fragments were removed. Also, since we are interested only on the raw sequence, proteins with the same sequence string have been collapsed in one entry. This was possible thanks

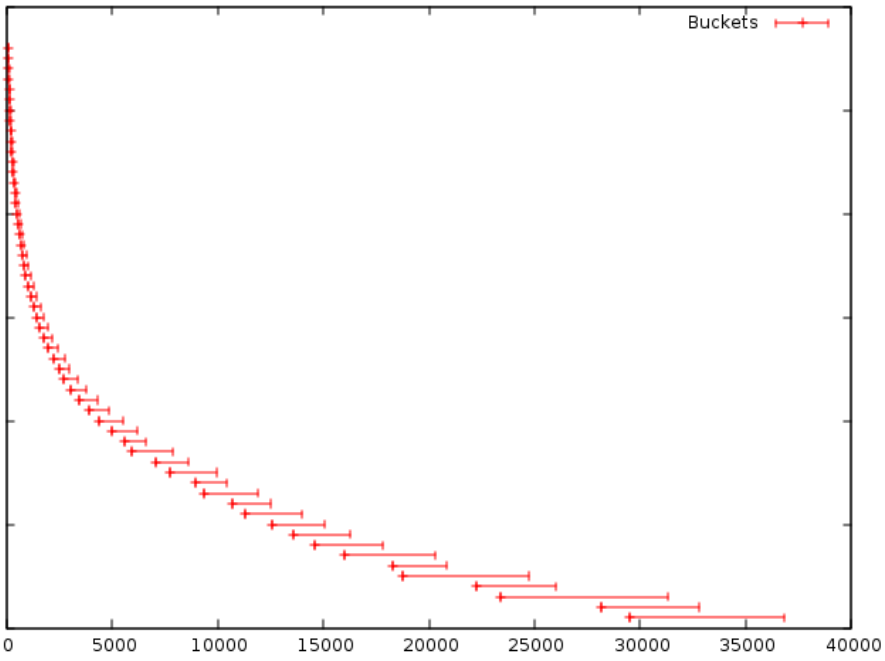

Figure 4.2: Best partitioning for sequences in Uniprot 2014_11.

to the hashing provided by the seguid method of the library Biopython. The lengths considered ranged from 100 up to 36 805. The grand total of sequences considered was 27 985 020. The distribution of sequences per length is shown in figure 4.1.

The cost of comparing each sequence with all the others, computed using equation 4.12, is about 3.1×10^{17}.

After computing for 10 minutes on a server equipped with 48 core and 256 Gigabytes of RAM, the algorithm identified a best partitioning solution composed by 56 subsets. Using the cost function defined by equation 4.11, this partitioning led to a total cost of about 5.3×10^{16}. The increase in performance is of six times, with a cost for the partitioning of about 16% of the full one.

The intervals in the evaluated partitioning are represented in figure 4.2.

This would be helpful in updating the system presented in chapter 5, since the time required to obtain the new edges is going to be quite smaller. Also, in updating the dataset, it is not required to perform again all the comparisons, but only to align the new sequences (query

dataset) to the whole set (reference database). The partitioning depends on the distribution of sequences in the reference database, since it could be supposed that the new sequences follow the same length distribution. The sequences in the query dataset can be then aligned only against the sequences in the intervals they belong to, achieving an even smaller computational cost.

The improvement does not depend on the number of sequences in the dataset, but only on their distribution. This is shown in table 4.2, where the number of sequences per each length was artificially increased and decreased to analyze possible variations on the cost.

Scale	Full cost	Minimum cost	Buckets	Speedup
$0,0625$	1.2×10^{15}	2.0×10^{14}	41	5.97
0.125	4.8×10^{15}	8.0×10^{14}	43	6.00
0.25	1.9×10^{16}	3.19×10^{15}	45	6.02
0.5	7.7×10^{16}	1.28×10^{16}	50	6.04
1.0	3.1×10^{17}	5.11×10^{16}	56	6.05
2.0	1.2×10^{18}	2.04×10^{17}	56	6.05
4.0	4.9×10^{18}	8.18×10^{17}	56	6.05
8.0	2.0×10^{19}	3.27×10^{18}	56	6.05

Table 4.2: Comparison of alignment costs given different sizes of the dataset. The first column represent the scale factor.

The reduced size in the reference database is also useful for transfer purposes: as stated before (section 4.1), most of the problems in aligning came from failures in transferring the reference database due to its size. Restricting the reference database to just the intervals identified in the partitioning leads to smaller files and then to a less probable failure in the transfers.

This approach is going to be tested in early 2015, to see if the better alignment performance could be enough to balance the growth rate in UniprotKB.

4.4 Alternative solutions

4.4.1 MPI Blast

Another solution explored was the usage of supercomputers like the IBM Blue Gene/Q available at the CINECA consortium[3]. The Blue Gene/Q infrastructure is a many core system relying on the Message Passing Interface (MPI) to coordinate jobs among the processors. However, the MPI implementation of Blast is lacking support and updates since 2010. On its website, developers suggest to use a commercial re-implementation of the software, called AbokiaBLAST, however Abokia website seemed not up-to-date too. Since dependencies of libraries and underlying MPI API could not be investigated before submitting a grant request for the usage of the Blue Gene/Q at CINECA, this option was left as a backup plan.

[3]http://www.cineca.it

Chapter 5

Graph databases

The huge amount of data representing connections between entities, be they social network data like the one collected by Facebook[1], interaction data like followers and re-tweets in Twitter[2] or the layout and management of a network infrastructure, highlighted the need of a more flexible and responsive database management system.

Classic and well tested DataBase Management Systems (DBMS) rely on the relational model presented by Codd between 1969 and 1970 [24]. However, relational databases and the usage of the Structured Query Language (SQL) for both definition and query limits the ability of running complex queries about connected data with a lot of hops between entities. This has been tested by the creators of Neo4J (see section 5.2.1), a popular graph database. As reported in [60], in a test database representing friendship connections, querying to get the friends-of-friends up to distance 5 from the origin takes quite a lot of time using traditional RDBMS and SQL. Using a graph database, it scales way better, leading to an almost real-time response, as shown in table 5.1

[1] http://www.facebook.com
[2] http://www.twitter.com

Depth	RDBMS	Neo4J
2	0.016	0.01
3	39.267	0.168
4	1543.505	1.359
5	Unfinished	2.132

Table 5.1: Comparison of RDBMS and Neo4J execution time for queries for friend-of-friend. Execution times are in seconds. Credits [60]

5.1 The technology

In a usual RDBMS implementation of a domain, data is stored inside tables composed by rows. Each row contains a field representing a different property of the data, like name, date of birth and address for a person. A field used to distinguish one row from another, like an identification number or a social security number, is called a key.

Tables can be joined usually using fields referencing keys from other tables, called foreign keys, leading to useful data. For example, it is possible to retrieve the list of all the items ordered by a specific customer, thanks to the reference to keys in the products table and the one in the clients table.

Graph databases are a rising technology used mostly to store and query connected data. Differently from standard RDBMS, data is not stored in tables and connected exploiting foreign keys. The domain is instead modelled using documents or key-value pairs, the vertices in the graph, and such elements are connected using direct references, or edges. In the previous example, all the items ordered by a customer could be reached by following the edges from the element representing the person to elements representing the items.

Different graph databases use different technologies. The most important aspects are the type of storage back-end and the query language used to navigate through the graph.

Two aspect of graph databases are worth mentioning The first one is the system used ad back-end to store the data. Neo4j (section 5.2.1) and OrientDB (section 5.2.2) use a native storage system, tightly coupling their inner data structure to the interrogation system. Titan (section 5.2.3) uses instead Cassandra [35] as storage, leaving the optimisation and all the data

management aspects to that technology.

The second element is the query language used to interrogate the database. Neo4J uses the native Cypher language, with queries resembling SQL in their structure. Titan, on the other end, uses only primitives from the Thinkerpop Blueprint stack, i.e. implementations of a specific Application Programming Interface (API). Details on both the systems are presented in their respective sub-sections of section 5.2.

5.1.1 Application in bioinformatics

The usage of graph databases in bioinformatics is not yet common. The fact that many biological domains can be modelled as graphs, for example the STRING pathway database [31], is a huge pro towards the adoption of this technology. One drawback in switching towards graph database is the fact that RDBMS are well studied and have a a number of optimisation techniques that outperform a poorly designed graph database in terms of query speed and responsiveness.

Anyways, in the last couple of years there have been some experiments with graph database in bioinformatics. A performance comparison in accessing data from the STRING database showed that the graph database technology can speed-up operations up to four orders of magnitude [34]. The project Bio4j [47], developed simultaneously as the one presented in this dissertation, mapped databases like Gene Ontology [4], UniprotKB and the NCBI Taxonomy to a graph. Their implementation is currently based on Neo4j technology (see section 5.2.1) and during FOSDEM 2014[3] they announced plans for versions using other technologies presented in the following section.

The main difference between Bio4j and the work presented in this chapter is that the underlying connections in Bio4j are already present in public databases, like relationship between Gene Ontology terms and annotated sequences. Then, besides the faster access, it is not different in results from the direct access to the original databases. The work presented in the remainder

[3]See https://archive.fosdem.org/2014/

of this chapter has instead a different perspective: besides the already known connections, a new set of relationship is built among vertices in the graph. The first test case is the set of relationship behind BAR+ (see section 2.4) and then the results from community detection algorithms presented in chapter 3.

5.2 Frameworks

5.2.1 Neo4j

Neo4j[4] is probably the first commercial graph database released. Developed by Neo Technologies, the first release of Neo4j dates back to 2010, while the most recent at the time of writing is version 2.1.6 of November 2014.

Neo4j authors "wrote the book" ([60]) about graph databases. Their effort in building a new technology to represent connected data was a natural response to the increasing amount of relationships between different elements of the models rising in the early years of the 21th century.

Neo4j uses a dedicated query language, Cypher, that has a syntax with a structure similar to SQL. The storage back-end was also developed internally by Neo Technologies, leading to a self-contained framework. The storage system is defined as a pure graph database, in contrast to other solutions like key-value stores and document-oriented databases.

5.2.2 OrientDB

OrientDB[5] is an open source graph database developed by Orient Technologies. Like Titan, it supports the Tinkerpop stack, but it is also possible to query the database using SQL. SQL queries are translated to OrientDB query system on the fly, in a transparent way to the user.

[4]available at http://neo4j.com/
[5]http://www.orientechnologies.com/orientdb/

OrientDB is advertised as available for free in its "Community edition" version but, at the time of testing for the purposes of this work, the download was not directly accessible: a subscription to OrientDB mailing list or a tweet on Twitter supporting the product was mandatory. The available version was also not clear: the latest one containing critical bug fixes and improvements was often available only via specific URL, not visible on the product Website.

OrientDB was the first graph database tested for the purposes of this work. The claimed openness was considered positive, since it was suggesting the ability to look at the inner workings of the system and develop customised solutions. The SQL support was also interesting from the point of view of future users lacking the programming skills for complex queries via Gremlin or just already familiar with the SQL syntax.

Tests however showed bad documentation for the system, leading to many problems in deploying a working server and uploading the data. Some of these problems have been solved exploiting the support system, however that showed how most of the solutions made use of undocumented features. Also, the support system was quite erratic, since the firm monetise the product via direct support in deployment and customisation.

Since the cons were more pressing that the pros, OrientDB was abandoned after developing a test case.

5.2.3 Titan

Titan[6] is a graph database that combines and exploits different technologies. The first release of Titan was version 0.1.0 in 2012, while the most recent release at the time of writing is version 0.5.2 of November 2014. The versions tested for this work were release 0.5.0 (August 2014) and 0.5.1 (October 2014). Version 0.4.4 (April 2014) was also tested, but the changes from version 0.4.x to 0.5.x were so deep that a re-design of data representation and of the software needed to interrogate the system was mandatory.

[6] Available at https://thinkaurelius.github.io/titan/

The storage back-end supported by Titan are Apache Cassandra[7] [35], Apache HBase[8] and Oracle Berkeley DB[9]. Cassandra is used by companies like Comcast, Hulu and Netflix.

Titan implements the TinkerPop graph stack[10], a framework developed with the help of different vendors in order to have a more agnostic framework. Titan also exploits technologies like the Hadoop[11] framework for distributed storage and computation, mostly known for its implementation of the map-reduce technology [27], and also search technologies like ElasticSearch for full text queries.

The default query language for Titan is Gremlin, also part of the TinkerPop stack. In Titan, the Groovy implementation of Gremlin is the most supported one, but also a Java implementation is available. Gremlin is quite straightforward in its for, exploiting both object oriented features like class methods and easy to understand terms.

Another part of TinkerPop implemented in Titan is the Rexster graph server, a Web-based server that allows the interrogation of the underlying graph using a Representational State Transfer (REST) Application Programming Interface.

5.2.4 Selected framework

Taking different aspects into account, the selected framework was Titan. One element was the possibility to scale the storage to different machines in a distributed environment. Another one was the easy to use query language, easy to understand also for biologists and bioinformaticians without a strong computer science background.

Another element in the decision was the work by McColl et al. [40], that showed a smaller memory usage for Titan compared to both Neo4j and OrientDB. Those two have been reported to be unfeasible on managing a graph with 128 million edges on a machine with 256 Gigabytes

[7] Available at https://cassandra.apache.org/
[8] Available at https://hbase.apache.org/
[9] Available at http://www.oracle.com/technetwork/database/berkeleydb
[10] See http://www.tinkerpop.com/
[11] Available at http://hadoop.apache.org

of memory. Given that the main dataset generated to replace BAR+ was composed by over 3 billion edges, Titan seemed an appropriate choice.

Still McColl et al. [40] report a better time performance of Titan in almost every case scenario. The only case in which Titan had the worst performance with respect to the other frameworks was in updating the graph. However, since the update of the graph in the case of the new BAR+ would only be done once or twice a year, this setback has been considered not crucial from an usage standpoint.

5.3 Database layout

The graph designed for representing the model was composed by different types of vertices and edges. Each vertex should convey information about a sequence or a property of a sequence, while edges should be used to traverse the graph in order to gather useful information.

The key types of vertices are described in the following sections, along to the edges they are involved in. More comprehensive lists of vertices planned to be added to the system and edges are described in section 5.3.6.

5.3.1 Sequence

A sequence vertex represent the raw string of text, i.e. the representation of its amino acid sequence, defining a protein. Since a protein sequence should bring its properties no matter the organism and the identification string attached to it, this was designed as the most important element of the graph.

Vertices of this class are identified by an hash evaluated on the raw amino acid sequence representation, using the seguid function provided by BioPython [19, 22]. Other useful data associated to the vertex is of course the sequence string and its length, so that sequences with a specific length can be searched without recomputing that value every time.

Sequence vertices are connected one another if there is a similarity relationship as described in the previous chapters. This type of edge is critical to identify clusters, communities and to connect distantly related protein sequences.

5.3.2 Sequence label

Each sequence vertex should be associated to a sequence label, i.e. an identifier in a repository like UniprotKB. Searching for Uniprot Accession or other identifiers in public databases is a core feature for databases like the one presented in this thesis. Different labels may refer to the same raw sequence, due to the fact that the same protein was found in different organisms, tissues or more simply since different databases use different identifiers.

Having an identification from public repositories may also be useful to associate the sequence to essays and other documentation stating the technology or other data regarding its discovery.

5.3.3 Annotation

Since the annotation of protein sequences with unknown functions is the key feature of both the original BAR+ (see chapter 2) and of the community detection algorithms presented in chapter 3, annotations should have their own class of vertices.

There vertices will be associated to sequence labels, alongside to the type of evidence of the annotation: an experimental evidence would be treated differently from an annotation inferred from sequence similarity.

The annotations themselves should also be connected one another, for example for representing the directed acyclic graph of the Gene Ontology. Such paths allows the identification of common ancestors in ontologies, related terms to identify cause-effect events and so on.

5.3.4 Organism

Like annotations, organism are layered in a structure, usually a taxonomy. Being able to reach all the sequences belonging to a specific organism, the clusters containing them and also identify the organisms associated to a specific community is another key aspect of the system.

As pointed out in section 2.4.1, there are many interests in specific organism like cattle and agriculture due to the resistance to parasites and increased productivity, or humans thanks to our very need of understanding diseases and design new drugs.

5.3.5 Structures

Three-dimensional structures, like the ones stored into the Protein Data Bank (PDB) [7], are the most important sources of information about a protein. Being able to retrieve all the PDB structures associated to a cluster or to a specific set of proteins may lead to insight about interactions of such proteins. Also, very often protein build complexes, i.e. a bigger functional unit composed by elements deriving from different sequences. Edges connecting these building blocks could connect also different clusters. Structures are also used to create profiles of families of proteins, then having a way to collect all the structures relative to a specific query would be useful.

5.3.6 Summary of vertices and edges

Summarising, the type of vertices already devised are:

- sequence: a string of residues
- sequence label: a sequence identifier in UniprotKB, Ensembl or another system
- annotation: something associated to a label, be it a GO term, an EC number and so on
- organism: an organism where a label was isolated on

- gene: a gene that can express a protein
- structure: a 3D representation of the protein structure
- disease: an illness, malfunction or abnormal behaviour related to a protein or gene
- cluster: a BAR+ specific entity used to enclose groups of similar sequences
- community: a community of sequences, as described in chapter 3

The types of edges are quite different in the relationship they can represent and the type of vertices they connect:

- similarity: a one-to-one sequence to sequence edge representing a similarity between the two sequences
- labelling: sequence to label edge, one-to-many, since the same sequence may have different identifiers
- belonging: label to organism, sequence to cluster
- annotating: label to annotation, one-to-many, specifying the known properties of the protein sequence along with qualities of the annotation like whether it has experimental evidence or not
- ontologies: a hierarchical representation of relationship between annotations, like a *is a* relationship between GO terms
- validation: cluster to annotation, identifying a statistically validated term for all the sequences in a cluster

5.4 Usage scenario

There are different possible usages for the system. The main purpose of BAR+ was to give statistically validated annotations for a sequence provided by the user. This is still possible given that the original technique is still in place.

5.4.1 Possible usage queries

The main concept behind using graph layout was to simplify the retrieval of information form the database. Different usage queries have been devised, but this list should be considered just as an example limited by the data currently planned for insertion. As long as more and different data would be added to the system, more informative and more complex queries would arise.

The first and most obvious query is the identification of connected components, i.e. clusters. Just following the similarity edges between sequence vertices, using well known algorithms like breadth-first search will lead to the identification of clusters. This kind of query falls exactly into the friend-of-friend example described at the beginning of this chapter, with similarity replacing friendship.

Another possible and most interesting usage is to navigate the graph in order to identify patterns or shared properties among elements in the graph. Starting from a 3D structure identified by a PDB, it would be possible to follow the edges leading to different labels, to the sequences, then to the possibly different clusters the sequences belong so that cross cluster properties could be highlighted.

The clustering procedure and the community detection not only partition the sequences in more coherent subsets, but also allow the reaching of different organisms and annotations. Is a community in a cluster specific to an organism or to an entry in the taxonomy? Are annotations shared by different clusters containing sequences of a taxonomical entry? Are he layers of EC numbers mapped to different resolutions in the community structure?

By starting from a well known sequence associated to a disease, queries may highlight new sequences that are potentially involved in similar negative effects.

5.4.2 Update

After adding new similarity edges between sequences evaluated using the approach described in chapter 4, the clusters and their statistical validation should be recomputed.

Clusters could be merged thanks to the execution of a breadth first search starting from the new vertices and collecting the cluster identifiers associated to the visited vertices. Splitting of clusters due to the deletion of sequences no longer in the database is also possible, thanks to the availability of the neighbours of the removed vertex. Just evaluating the connected components on the neighbourhood of the newly added or deleted vertices would be enough.

Annotations could be collected starting from the cluster vertex, then moving to the sequences, to their labels and to the annotation vertex, or in the opposite direction, starting from the annotation. Besides the direct annotation of the sequences, hierarchical annotations like ancestors of a GO term could be retrieved thanks to the annotation-to-annotation relationships.

As reported in section 5.2.4, adding and removing vertices and edges may be the most computationally heavy procedure, while visiting the graph to evaluate the new statistically validated annotations is expected to be fast.

New annotations could be added in a similar way. Not only newly discovered relationships between sequences and known GO terms or between sequences and recently crystallised structures could be added, but also whole new annotation systems can expand the graph. Since vertices can have arbitrary types, called labels in Titan, adding a new one does not require complex design decisions about legacy structures: a new annotation type needs to be defined and edges between those vertices and the sequence labels (or sequences directly) need to be established.

Statistical validation can be performed only on the new annotation type. Legacy queries are not being affected if they required a specific annotation type. Otherwise, an old query requesting annotations of any type would now give new results and insights in a transparent way.

5.5 Deployment

At the time of writing, the system is still being built. This was due to some setbacks in the process, mostly hardware failures. If no other problems arise, the system should be up and working before the thesis defense. However, since the analysis of such amount of data requires

time, only a subset of the devised experiments may be ready.

To manage the amount of data, a dedicated data server was deployed, with an initial storage capacity of about one Terabyte. That server is connected to the computing server, a system with 48 processing units and 256 Gigabytes of RAM, via a 10 Gigabit/s Ethernet connection.

The computing server hosts the data analysis software, developed using the already mentioned graph-tool library (see section 3.5.4), and the Titan server. More supporting software was developed in Java, thanks to the API available in Titan.

Chapter 6

Conclusion

6.1 Summary of Thesis Achievements

Automated annotation of protein sequences is a core task in managing the ever growing amount of biomedical data. In this thesis an already proven technique is improved thanks to the implementation of community detection techniques. Community detection showed an increased ability in isolate very specific annotations inside a bigger set, leading to a more focused attribution of them to specific protein sequences. A fast algorithm for community detection was identified and tested, while improvements to software libraries for graph manipulations have been proposed.

The usage of graph databases for storing and querying the connected data, both sequences and annotations, was tested and proposed as new solution for such tasks. The faster access and traversal of graphs using this technology compared to the usage of multiple joins in SQL-based relational database is shown. A specific graph database technology has been deployed and tested, to assess its feasibility.

6.2 Future Work

A complete and well studied system needs to be completed. The technical challenges showed that a solid state memory may be more feasible to batch uploading of the data, thanks to the faster access. A distributed storage may also be a solution, due to the ability of Cassandra to scale.

The implementation of the Louvain method for community detection in the graph-tool library needs to be checked before submitting it for vetting to the author of the library. Different approaches and ensembles of algorithms should be tested to improve the accuracy of the system.

The partitioning algorithm for the dataset, presented in chapter 4 was submitted to a peer-reviewed journal, the acceptance is pending. The implementation of the algorithm will be released to the public before the publication of the results.

Automated annotations of proteins arising from the new system are going to be tested in the upcoming CAFA challenge, which should take place at the beginning of 2016.

Bibliography

[1] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[2] Christophe Anjard, William F Loomis, et al. Evolutionary analyses of abc transporters of dictyostelium discoideum. *Eukaryotic Cell*, 1(4):643–652, 2002.

[3] Jac M Anthonisse. The rush in a directed graph. *Stichting Mathematisch Centrum. Mathematische Besliskunde*, (BN 9/71):1–10, 1971.

[4] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, May 2000.

[5] L. Bartoli, L. Montanucci, R. Fronza, P.L. Martelli, P. Fariselli, L. Carota, G. Donvito, G.P. Maggi, and R. Casadio. The bologna annotation resource: a non hierarchical method for the functional and structural annotation of protein sequences relying on a comparative large-scale genome analysis. *Journal of Proteome Research*, 8(9):4362–4371, 2009.

[6] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. 2009.

[7] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, January 2000.

[8] Ronnie P-A Berntsson, Sander HJ Smits, Lutz Schmitt, Dirk-Jan Slotboom, and Bert Poolman. A structural classification of substrate-binding proteins. *FEBS letters*, 584(12):2606–2617, 2010.

[9] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, October 2008.

[10] B. Boeckmann, A. Bairoch, R. Apweiler, M.C. Blatter, A. Estreicher, E. Gasteiger, M.J. Martin, K. Michoud, C. O'Donovan, I. Phan, et al. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic acids research*, 31(1):365–370, 2003.

[11] P. Boldi and S. Vigna. The webgraph framework i: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 595–602, New York, NY, USA, 2004. ACM.

[12] Paolo Boldi and Sebastiano Vigna. The webgraph framework II: codes for the world-wide web. In *Data Compression Conference, 2004. Proceedings. DCC 2004*, pages 528–, March 2004.

[13] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30(2):136–145, 2008.

[14] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 20(2):172–188, 2008.

[15] U. Brandes, D. Delling, M. Gaertler, R. Grke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. *Arxiv preprint physics/0608255*, 2006.

[16] Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[17] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Grke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On finding graph clusterings with maximum modularity. In Andreas Brandstdt, Dieter Kratsch, and Haiko Mller, editors, *Graph-Theoretic Concepts in Computer Science*, number 4769 in Lecture Notes in Computer Science, pages 121–132. Springer Berlin Heidelberg, January 2007.

[18] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007.

[19] Brad Chapman and Jeffrey Chang. Biopython: Python tools for computational biology. *SIGBIO Newsl.*, 20(2):15–19, August 2000.

[20] Wyatt T. Clark and Predrag Radivojac. Information-theoretic evaluation of predicted ontological annotations. *Bioinformatics*, 29(13):i53–i61, July 2013.

[21] A. Clauset, M.E.J. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[22] Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. Biopython: freely available python tools for computational molecular biology and. *Bioinformatics*, 25(11):1422–1423, June 2009.

[23] Peter JA Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, et al. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009.

[24] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[25] The UniProt Consortium. Ongoing and future developments at the universal protein resource. *Nucleic Acids Research*, 39(suppl 1):D214–D219, January 2011.

[26] Amy L Davidson, Elie Dassa, Cedric Orelle, and Jue Chen. Structure, function, and evolution of bacterial atp-binding cassette systems. *Microbiology and Molecular Biology Reviews*, 72(2):317–364, 2008.

[27] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[28] Robert D. Finn, Jaina Mistry, John Tate, Penny Coggill, Andreas Heger, Joanne E. Pollington, O. Luke Gavin, Prasad Gunasekaran, Goran Ceric, Kristoffer Forslund, Liisa Holm, Erik L. L. Sonnhammer, Sean R. Eddy, and Alex Bateman. The pfam protein families database. *Nucleic Acids Research*, 38(suppl 1):D211–D222, January 2010.

[29] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(35):75–174, February 2010.

[30] Santo Fortunato and Marc Barthlemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, January 2007.

[31] Andrea Franceschini, Damian Szklarczyk, Sune Frankild, Michael Kuhn, Milan Simonovic, Alexander Roth, Jianyi Lin, Pablo Minguez, Peer Bork, Christian von Mering, et al. STRING v9. 1: protein-protein interaction networks, with increased coverage and integration. *Nucleic acids research*, 41(D1):D808–D815, 2013.

[32] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[33] M. Girvan and M.E.J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821, 2002.

[34] Christian Theil Have and Lars Juhl Jensen. Are graph databases ready for bioinformatics? *Bioinformatics*, 29(24):3107–3108, December 2013.

[35] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[36] Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80(5):056117, November 2009.

[37] Andrea Lancichinetti and Santo Fortunato. Limits of modularity maximization in community detection. *Physical Review E*, 84(6):066122, December 2011.

[38] Andrea Lancichinetti, Filippo Radicchi, Jos J. Ramasco, and Santo Fortunato. Finding statistically significant communities in networks. *PLoS ONE*, 6(4):e18961, April 2011.

[39] Arthur Lesk. *Introduction to Bioinformatics*. OUP Oxford, March 2008.

[40] Robert Campbell McColl, David Ediger, Jason Poovey, Dan Campbell, and David A. Bader. A performance evaluation of open source graph databases. In *Proceedings of the First Workshop on Parallel Programming for Analytics Applications*, PPAA '14, pages 11–18, New York, NY, USA, 2014. ACM.

[41] S. McGinnis and T.L. Madden. BLAST: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic acids research*, 32(suppl 2):W20–W25, 2004.

[42] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP'07, pages 529–543, Berlin, Heidelberg, 2007. Springer-Verlag.

[43] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, February 2004.

[44] M.E.J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):036104, 2006.

[45] M.E.J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577, 2006.

[46] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: bringing order to the web. 1999.

[47] Pablo Pareja-Tobes, Eduardo Pareja-Tobes, Marina Manrique, Eduardo Pareja, and Raquel Tobes. Bio4j: An open source biological data integration platform. In *IWBBIO*, page 281, 2013.

[48] Tiago P. Peixoto. The graph-tool python library. *figshare*, 2014.

[49] Catia Pesquita, Daniel Faria, André O. Falcão, Phillip Lord, and Francisco M. Couto. Semantic similarity in biomedical ontologies. *PLoS Comput Biol*, 5(7):e1000443, July 2009.

[50] Damiano Piovesan, Pier Luigi Martelli, Piero Fariselli, Giuseppe Profiti, Andrea Zauli, Ivan Rossi, and Rita Casadio. How to inherit statistically validated annotation within BAR+ protein clusters. *BMC Bioinformatics*, 14(Suppl 3):S4, 2013.

[51] Damiano Piovesan, Pier Luigi Martelli, Piero Fariselli, Andrea Zauli, Ivan Rossi, and Rita Casadio. BAR-PLUS: the bologna annotation resource plus for functional and structural annotation of protein sequences. *Nucleic Acids Research*, 39:W197–W202, May 2011.

[52] Damiano Piovesan, Giuseppe Profiti, Pier Luigi Martelli, and Rita Casadio. The human "magnesome": detecting magnesium binding sites on human proteins. *BMC Bioinformatics*, 13(Suppl 14):S10, 2012.

[53] Damiano Piovesan, Giuseppe Profiti, Pier Luigi Martelli, Piero Fariselli, and Rita Casadio. Extended and robust protein sequence annotation over conservative non hierarchical clusters: the case study of the ABC transporters. *Emerging Technologies in Computing Systems, ACM Journal on*, 9(4):27:1–27:8, November 2013.

[54] Damiano Piovesan, Giuseppe Profiti, Pier Luigi Martelli, Piero Fariselli, Luca Fontanesi, and Rita Casadio. SUS-BAR: a database of pig proteins with statistically validated structural and functional annotation. *Database*, 2013:bat065, 2013.

[55] Giuseppe Profiti, Pier Luigi Martelli, and Rita Casadio. A pipeline for predicting the function of the AFP/CAFA 2014 targets at the Bologna Biocomputing Group. In *22nd Annual Conference on Intelligent Systems for Molecular Biology Automated Function Prediction Special Interest Group*, 2014.

[56] Giuseppe Profiti, Damiano Piovesan, Pier Luigi Martelli, Piero Fariselli, and Rita Casadio. Community detection within clusters helps large scale protein annotation - preliminary results of modularity maximization for the bar+ database. In *4th International Conference on Bioinformatics Models, Methods and Algorithms*, pages 328–332, 2013.

[57] Giuseppe Profiti, Damiano Piovesan, Pier Luigi Martelli, Piero Fariselli, and Rita Casadio. Protein sequence annotation by means of community detection. In *IWBBIO 2013*, pages 753–755, 2013.

[58] Giuseppe Profiti, Damiano Piovesan, Pier Luigi Martelli, Piero Fariselli, and Rita Casadio. Protein sequence annotation by means of community detection. *Current Bioinformatics*, 10(2):to appear, 2015.

[59] Predrag Radivojac, Wyatt T Clark, Tal Ronnen Oron, Alexandra M Schnoes, Tobias Wittkop, Artem Sokolov, Kiley Graim, Christopher Funk, Karin Verspoor, Asa Ben-Hur, Gaurav Pandey, Jeffrey M Yunes, Ameet S Talwalkar, Susanna Repo, Michael L Souza, Damiano Piovesan, Rita Casadio, Zheng Wang, Jianlin Cheng, Hai Fang, Julian Gough, Patrik Koskinen, Petri Toronen, Jussi Nokso-Koivisto, Liisa Holm, Domenico Cozzetto, Daniel W A Buchan, Kevin Bryson, David T Jones, Bhakti Limaye, Harshal Inamdar, Avik Datta, Sunitha K Manjari, Rajendra Joshi, Meghana Chitale, Daisuke Kihara, Andreas M Lisewski, Serkan Erdin, Eric Venner, Olivier Lichtarge, Robert Rentzsch, Haixuan Yang, Alfonso E Romero, Prajwal Bhat, Alberto Paccanaro, Tobias Hamp, Rebecca Kaszner, Stefan Seemayer, Esmeralda Vicedo, Christian Schaefer, Dominik Achten, Florian Auer, Ariane Boehm, Tatjana Braun, Maximilian Hecht, Mark Heron, Peter Honigschmid, Thomas A Hopf, Stefanie Kaufmann, Michael Kiening, Denis Krompass, Cedric Landerer, Yannick Mahlich, Manfred Roos, Jari Bjorne, Tapio Salakoski, Andrew Wong, Hagit Shatkay, Fanny Gatzmann, Ingolf Sommer, Mark N Wass, Michael J E Sternberg, Nives

Skunca, Fran Supek, Matko Bosnjak, Pance Panov, Saso Dzeroski, Tomislav Smuc, Yiannis A I Kourmpetis, Aalt D J van Dijk, Cajo J F ter Braak, Yuanpeng Zhou, Qingtian Gong, Xinran Dong, Weidong Tian, Marco Falda, Paolo Fontana, Enrico Lavezzo, Barbara Di Camillo, Stefano Toppo, Liang Lan, Nemanja Djuric, Yuhong Guo, Slobodan Vucetic, Amos Bairoch, Michal Linial, Patricia C Babbitt, Steven E Brenner, Christine Orengo, Burkhard Rost, Sean D Mooney, and Iddo Friedberg. A large-scale evaluation of computational protein function prediction. *Nature Methods*, 10(3):221–227, March 2013.

[60] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. "O'Reilly Media, Inc.", June 2013.

[61] Milton H. Saier, Can V. Tran, and Ravi D. Barabote. TCDB: the transporter classification database for membrane transport protein analyses and information. *Nucleic Acids Research*, 34(suppl 1):D181–D186, January 2006.

[62] Jason E Stajich, David Block, Kris Boulez, Steven E Brenner, Stephen A Chervitz, Chris Dagdigian, Georg Fuellen, James GR Gilbert, Ian Korf, Hilmar Lapp, et al. The bioperl toolkit: Perl modules for the life sciences. *Genome research*, 12(10):1611–1618, 2002.

[63] Baris E. Suzek, Hongzhan Huang, Peter McGarvey, Raja Mazumder, and Cathy H. Wu. UniRef: comprehensive and non-redundant UniProt reference clusters. *Bioinformatics*, 23(10):1282–1288, May 2007.

[64] Koen Wagner, Geri F Moolenaar, and Nora Goosen. Role of the two atpase domains of escherichia coli uvra in binding non-bulky dna lesions and interaction with uvrb. *DNA repair*, 9(11):1176–1186, 2010.

[65] Edwin C Webb et al. *Enzyme nomenclature 1992. Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology on the Nomenclature and Classification of Enzymes.* Number Ed. 6. Academic Press, 1992.

[66] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, pages 452–473, 1977.