5-2011

Graph Kernels and Applications in Bioinformatics

Marco Alvarez Vega
Utah State University

GRAPH KERNELS AND APPLICATIONS IN BIOINFORMATICS

by

Marco Alvarez Vega

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

Dr. Xiaojun Qi Major Professor	Dr. Changhui Yan Committee Member
Dr. Minghui Jiang Committee Member	Dr. Adele Cutler Committee Member
Dr. Vicki Allan Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2011

Abstract

Graph Kernels and Applications in Bioinformatics

by

Marco Alvarez Vega, Doctor of Philosophy

Utah State University, 2011

Major Professor: Dr. Xiaojun Qi
Department: Computer Science

In recent years, machine learning has emerged as an important discipline. However, despite the popularity of machine learning techniques, data in the form of discrete structures are not fully exploited. For example, when data appear as graphs, the common choice is the transformation of such structures into feature vectors. This procedure, though convenient, does not always effectively capture topological relationships inherent to the data; therefore, the power of the learning process may be insufficient. In this context, the use of kernel functions for graphs arises as an attractive way to deal with such structured objects.

On the other hand, several entities in computational biology applications, such as gene products or proteins, may be naturally represented by graphs. Hence, the demanding need for algorithms that can deal with structured data poses the question of whether the use of kernels for graphs can outperform existing methods to solve specific computational biology problems. In this dissertation, we address the challenges involved in solving two specific problems in computational biology, in which the data are represented by graphs.

First, we propose a novel approach for protein function prediction by modeling proteins as graphs. For each of the vertices in a protein graph, we propose the calculation of evolutionary profiles, which are derived from multiple sequence alignments from the amino acid residues within each vertex. We then use a shortest path graph kernel in conjunction

with a support vector machine to predict protein function. We evaluate our approach under two instances of protein function prediction, namely, the discrimination of proteins as enzymes, and the recognition of DNA binding proteins. In both cases, our proposed approach achieves better prediction performance than existing methods.

Second, we propose two novel semantic similarity measures for proteins based on the gene ontology. The first measure directly works on the gene ontology by combining the pairwise semantic similarity scores between sets of annotating terms for a pair of input proteins. The second measure estimates protein semantic similarity using a shortest path graph kernel to take advantage of the rich semantic knowledge contained within ontologies. Our comparison with other methods shows that our proposed semantic similarity measures are highly competitive and the latter one outperforms state-of-the-art methods. Furthermore, our two methods are intrinsic to the gene ontology, in the sense that they do not rely on external sources to calculate similarities.

(109 pages)

Public Abstract

Graph Kernels and Applications in Bioinformatics

Nowadays, machine learning techniques are widely used for extracting knowledge from data in a large number of bioinformatics problems. It turns out that in many of such problems, data observations can be naturally represented by discrete structures such as graphs, networks, trees, or sequences. For example, a protein can be seen as a cloud of interconnected atoms lying on a 3-dimensional space. The focus of this dissertation is on the development and application of machine learning techniques to bioinformatics problems wherein the data can be represented by graphs. In particular, we focus our attention on proteins, which are essential elements in the life process. The study of their underlying structure and function is one of the most important subjects in bioinformatics. As proteins can be naturally represented by graphs, we consider the use of kernel functions that can directly deal with data observations in the form of graphs. Kernel functions are the basic building block for a powerful family of machine learning algorithms called kernel methods.

Concretely, we propose a novel approach for predicting the function of proteins. We model proteins as graphs, and we predict function using support vector machines and graph kernels. We evaluate our approach under two types of function prediction, the discrimination of proteins as enzymes or not, and the recognition of DNA binding proteins. In both cases, the resulting performance is higher than existing methods.

In addition, given the establishment of ontologies as a popular topic in biomedical research, we propose two novel semantic similarity measures between pairs of proteins. First, we introduce a novel semantic similarity method between pairs of gene ontology terms. Second, we propose an instance of the shortest path graph kernel for calculating the semantic similarity between proteins. This latter approach, when compared with state-of-the-art methods, yields an improved performance.

To my parents

Acknowledgments

Many people have contributed in different ways to the work contained in this dissertation. I sincerely thank professors Xiaojun Qi, Changhui Yan, and Seungjin Lim. They have provided the required resources and supervision along the completion of this work. I am also greatly indebted to Dr. Xiaojun Qi for her encouragement and words of wisdom.

I would like to express my gratitude and appreciation to Dr. Adele Cutler for her valuable advice and motivation that have led me to pursue interesting research topics in machine learning. I am grateful to my committee members for having followed my work. I would also like to thank Myra Cook for her assistance with refining this dissertation.

I would like to acknowledge the Center for High Performance Computing at Utah State University for enabling the development of computationally intensive research activities. Thanks to their resources, I was able to reduce the running time of each experiment from several days to hours.

There are many individuals who have contributed to my academic career since I started my undergraduate studies. I am always grateful to all my former professors, especially Dr. Andre de Carvalho at the University of São Paulo and Dr. Nalvo de Almeida Jr at the Federal University of Mato Grosso do Sul for their continuous support. I should also acknowledge former colleagues at the Dom Bosco Catholic University, Dr. Milton Romero and Dr. Hemerson Pistori, for their encouragement, friendship, and recommendations for accomplishing my goals.

Finally, I want to express my love to my family and friends in Brazil, Peru, and Logan. My parents, sisters, “Tío Mario”, “Tías Vicky and Ruth”, Miguel and many other relatives and friends have played a huge role when adversity came around during this journey.

Marco

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In many real world problems, data observations occur naturally as complex structures, which in turn can be represented by powerful formalisms such as graphs. Generally speaking, a graph is defined as a collection of vertices or nodes where pairs of vertices are connected by edges. Graphs are increasingly being used to model structured data objects in more and more domains. For example, in social science, social networks are represented as finite graphs of ties between social actors [1]. In natural language processing, graphs are used to encode the meaning and structure of texts [2]. In chemoinformatics, graphs are the fundamental way for representing chemical compounds [3]. In bioinformatics, graphs and sequences are also used to represent the information contained in important bodies such as biomolecules, DNA, RNA, and proteins [4]. Note that from a computational perspective, sequences are in fact a special case of graphs.

Particularly in bioinformatics, recent scientific and technological advances have contributed to the production of enormous amounts of structured data, which are available in private and public repositories. This context has brought the attention of machine learning and data mining researchers, who are increasingly focusing on the study of patterns existing in such structures [5]. One case of great interest and importance is the study of protein function.

Proteins are essential macromolecules responsible for performing numerous functions in living cells. Understanding their function is crucial for promoting the advancement of knowledge in biology and life sciences in general. Each protein within the body has a specific function. For example, inside a cell we find a special type of protein, the enzyme, which is known to increase the rate of chemical reactions.

The structural information contained in proteins is very rich and can be represented

in distinct ways. Based on the sequence of amino acid residues on protein chains, we can represent proteins as sequences of letters with each letter representing one amino acid residue. Based on the structure of a protein, i.e., the spatial coordinates of each of the atoms constituting a protein, we can represent proteins as point clouds in a 3-dimensional space. Several structural genomics projects, whose goal is to determine the 3-dimensional structure of proteins, have been producing a vast number of protein structures. Furthermore, due to the difficulties involved in the experimental characterization of protein function, these projects have been generating a large number of proteins with little or unknown functional information [6]. Consequently, effective and high-throughput computational approaches are needed for addressing the problem of predicting the unknown function of proteins. In this dissertation, we investigate graph models for proteins that can effectively represent their rich structural information. We propose the use of such models in conjunction with graph kernels and support vector machines for predicting protein function in two specific problems: the prediction of proteins as enzymes and the prediction of proteins as DNA binding proteins.

In machine learning, kernel methods such as the widely used support vector machine, are a class of very powerful algorithms for pattern analysis and prediction [7–9]. A crucial point in kernel methods is that they depend on the data only through the calculation of kernel functions. Therefore, algorithms of the family of kernel methods are based upon the use of kernel functions (or simply "kernels"), which can be thought of as similarity measures for pairs of objects. In statistical learning theory, a kernel function $K(x, y)$ for a pair of data observations x and y is a formalism used to extend linear methods to work in a higher dimensional space where nonlinearities in the data can be detected. Several kernel functions are available in the literature, the most popular of which assume that the observations are feature vectors, i.e., arrays of numeric values. A list of some kernels defined to work on feature vectors include: linear, polynomial, gaussian, exponential, and laplacian kernels [9].

It turns out that we can also define kernel functions for pairs of discrete structures. More specifically, we can define kernel functions for graph models of proteins. Thus, the

entire family of kernel methods is available for building prediction models by directly processing datasets of graphs. This appears as an elegant and effective way for solving protein function prediction problems. Among all recent developments, graph kernels based on shortest paths [10], are a remarkable class of kernel functions. They are able to compare graphs at acceptable polynomial time. Contrary to most existing graph kernels, shortest path graph kernels can deal with graphs with continuous feature vectors at their nodes. In our approach, shortest path graph kernels are used in conjunction with support vector machines to train models for predicting protein function.

On the other hand, a totally different way to characterize proteins is the use of ontological annotations, that is, by labeling proteins with multiple terms defined in a related ontology. Ontologies are a popular topic in various fields of computer science [11]. They are primarily used to represent the knowledge within a domain, by means of a hierarchical structure of concepts connected by relationships between them. This formal specification permits the description of entities within the investigated domain, and eventually, the use of computational methods for reasoning about such entities. In bioinformatics, the gene ontology [12] is a major initiative to systematically organize biological knowledge across species and databases.

The gene ontology provides a controlled vocabulary of terms used to characterize gene products, either RNA or proteins, in terms of their associated biological processes, cellular components, and molecular functions. The vocabulary of terms is organized as a directed acyclic graph, where each term has defined relationships to one or more other terms. Given the vocabulary, proteins can be annotated with terms to describe their characteristics. A rich repository of protein annotations by gene ontology terms is produced by the GOA project [13] held at the European Bioinformatics Institute, which provides high-quality annotations to proteins in the UniProt KnowledgeBase. The extensive coverage of biological knowledge in the gene ontology in conjunction with the increasing number of protein annotations deposited in their respective databases, make these resources valuable components for the analysis of proteins.

For instance, proteins can be compared based on the semantic similarity between their respective annotating terms. The term semantic similarity is a measure for the likeness of the meaning of the involved terms. Therefore, protein semantic similarity can be derived by analyzing their respective annotation sets and the semantic relationships of such terms in the gene ontology. In this dissertation, we also propose graph models for proteins based on their respective annotating terms and relationships in the gene ontology. Given such models, we propose two novel ways to estimate the semantic similarity between pairs of proteins. The estimation of semantic similarity between a pair of proteins is a basic building block that can be incorporated in the study of protein function.

1.1 Contributions

In view of the abundance of structured data in bioinformatics, specifically, data provided by databases of protein structures, the gene ontology, and databases of annotated proteins, we focus on the study and proposal of computational approaches that can take advantage of the rich amount of information contained in proteins, when they are represented as structured objects. To this end, we consider two distinct problems in bioinformatics, namely, protein function prediction and protein semantic similarity.

1.1.1 Protein Function Prediction

We address the problem of protein function prediction by proposing a novel approach for modeling proteins as graphs. For a given protein, a graph model is created by combining information derived from the protein sequence and structure. We apply our approach to two separate binary classification problems, namely, the discrimination of proteins as enzymes, and the discrimination of proteins as DNA-binding proteins. In both cases, graph kernels and support vector machines are used to train the binary classifiers. In summary, our contributions for protein function prediction are highlighted below:

- We propose novel graph models for proteins. We analyze the 3-dimensional protein structure in order to create a protein graph, wherein the vertices represent groups

of neighboring amino acid residues, and edges connect pairs of vertices that are in contact, i.e., vertices whose their closest atoms are under a minimum distance defined as $5\mathring{A}$;

- We propose the calculation of evolutionary profiles for each of the vertices in a protein graph. These profiles are derived from multiple sequence alignments for the amino acid residues residing within the vertex. The motivation behind this idea is to characterize those regions in the protein structure that seem to be more conserved over time;

- We propose to apply our graph models to datasets of protein structures, so that, shortest path graph kernels can be used in conjunction with the support vector machine algorithm for binary classification in two distinct protein function problems. We show that our proposed approach outperforms other state-of-the-art methods in both problems, the discrimination of enzymes and the recognition of DNA binding proteins. The publications derived from this work are [14, 15].

1.1.2 Protein Semantic Similarity

We also focus our attention on the problem of calculating the semantic similarity between proteins. We propose two novel methods to estimate protein semantic similarity. In both methods, the similarity score for a pair of proteins is derived from their respective annotations from the gene ontology. The hierarchical structure of the gene ontology allows us to model proteins as graphs. For example, for a given protein, a graph model can be created by extracting a subgraph from the gene ontology, such that it contains all the terms (vertices) annotating the protein, along with their respective ancestors in the ontology. This subgraph also includes the relationships (edges) existing in the gene ontology between each pair of vertices. Once gene ontology terms and their relationships provide very valuable semantic knowledge that can be used for estimating functional similarities between proteins, we present our methods under the common denomination of protein semantic similarity. Our contributions for estimating the semantic similarity between pairs of proteins are highlighted below:

- We propose a semantic similarity method that can work directly on the gene ontology. For a pair of input proteins, their similarity is calculated by combining pairwise semantic similarity scores between their respective annotating terms. To this end, we estimate the semantic similarity between a pair of gene ontology terms, by taking into account the length of the shortest path between them, the depth of their nearest common ancestor, and the semantic similarity between the definitions accompanying both terms. To the best of our knowledge, this is the first attempt to use similarities between definitions of terms from the gene ontology to compute the semantic similarity. The publications directly associated with this method are [16–18];

- We propose a novel method for estimating protein semantic similarity. We estimate similarity by representing two input proteins as induced subgraphs from the gene ontology, and then applying an instance of the shortest path graph kernel. By using this kernel function, machine learning methods based on kernels can take advantage of the rich semantic knowledge contained within ontologies and be directly applied to datasets of proteins. The publication derived from this method is [19];

- We introduce the above two novel methods with the advantage that they are intrinsic to the gene ontology, in the sense that they do not rely on external sources to calculate similarities. Most of existing methods include scores calculated from external annotation databases; therefore, they are likely to be biased by proteins that are studied more intensively [20]. In a comprehensive evaluation using a benchmark dataset, both of our proposed methods compare favorably with other existing methods. Our approaches provide an alternative route, with comparable performance, to methods that use external resources.

1.2 Organization of This Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, we introduce basic concepts and terminology related to statistical learning theory and kernel methods.

Such elements are essential for understanding kernel functions for the graph domain. Chapter 3 reviews state-of-the-art kernels for graphs, which are the major ingredients for solving two computational biology problems: protein function prediction and protein semantic similarity. Chapter 4 describes our proposed graph models for protein structures and their evaluation under two different function prediction problems: the discrimination of proteins as enzymes and the prediction of DNA binding proteins. Chapter 5 reviews the problem of protein semantic similarity and describes our two proposed algorithms: SSA and the shortest path graph kernel, which are accompanied by their respective evaluation and comparison with state-of-the-art methods. Finally, Chapter 6 summarizes our contributions with their respective publications. It also suggests directions for future work.

Chapter 2

Statistical Learning and Kernel Methods

This chapter introduces the basic foundations of learning theory and kernel methods. They provide the formalisms required to explore and understand kernel methods in the graph domain and their application to computational biology problems.

The chapter starts by introducing the basic concepts and definitions supporting statistical learning theory. Then the formulation of support vector machines is described in detail in conjunction with the connection existing between risk minimization principles and learning with linear classifiers. Finally, kernels are presented as an attractive way to extend linear classifiers to nonlinear spaces.

As the protein function prediction problems described in this dissertation are classification problems involving only two classes, when possible, the examples and settings for the concepts described in this chapter are primarily concerned with and restricted to the problem of binary classification.

2.1 Statistical Learning Theory

Machine Learning can be informally viewed as the process of automatically learning from observed experience, so that models can be constructed and predictions can be made using such models. Statistical learning theory (SLT) [21] comprises the underlying theoretical framework for many existing algorithms for the problem of learning from examples [22]. Although the original motivation behind SLT was philosophical in nature, it provided the basis for new learning algorithms and gained popularity with the development of the well-known support vector machine (SVM) [23].

In the context of this dissertation, a special case of learning is considered, namely, *supervised learning*. Herein, the question of how a machine can *learn* specific tasks by

observing data is addressed by feeding the machine a set of known *examples*. Accordingly, a model that identifies regularities in the data can be inferred and subsequently used for future predictions on unseen examples.

Formally, the set of known observations, also called the *training set*, is the set of n pairs

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y} \tag{2.1}$$

where $\mathcal{X}$ is an *input space* of *instances* and $\mathcal{Y}$ is an *output space* of *labels*. Typically, $y_i \in \mathbb{R}$ for regression problems while y_i is discrete for classification problems. For example, consider the protein function prediction problems addressed in this dissertation. They are classification problems restricted to two classes, i.e., *binary classification*, where the output space can be defined as $\mathcal{Y} = \{0, 1\}$. In order to learn a model, the learning algorithm must find a mapping f, also called a *classifier*, from the space of functions $\mathcal{F}$, where $f \colon \mathcal{X} \to \mathcal{Y}$ makes as few misclassifications as possible. From this point on, this chapter focuses on the problem of binary classification.

Note that the process of learning a classifier does not make any assumptions on the nature of $\mathcal{X}$ or $\mathcal{Y}$. However, in order to make learning possible, we assume the existence of an unknown but fixed *joint probability distribution* $\mathcal{P}$ on $\mathcal{X} \times \mathcal{Y}$, which is the model governing the phenomenon of data generation. Within this model, both past observations used for training and future unseen examples are related by $\mathcal{P}$, because they are assumed to be generated by sampling independent and identically distributed (i.i.d.) random observations from the distribution $\mathcal{P}$.

2.1.1 Loss Functions and Risk

In summary, the goal of supervised learning consists of the following: given a training set $D = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}$ containing n i.i.d. observations sampled from $\mathcal{P}$, find a classifier f that can later be used with any $x \in \mathcal{X}$ to predict the corresponding $y \in \mathcal{Y}$. Certainly, it is unavoidable to have a measure of how well the classifier f is performing. More specifically, we need a *loss function* $l \colon \mathcal{X} \times \mathcal{Y} \times \mathcal{F} \to \mathbb{R}$ that measures the cost of classifying an input

example $x \in \mathcal{X}$ with the label $y \in \mathcal{Y}$. For example, the simple 0-1 loss function for classification is defined as:

$$l(x, y, f) = \begin{cases} 1, & \text{if } f(x) \neq y \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

While the loss function is used to measure the error at individual observations, the *risk* of a classifier f according to the distribution $\mathcal{P}$ is the expected loss, as given by:

$$R(f) = \int_{\mathcal{X} \times \mathcal{Y}} l(x, y, f) \cdot d\mathcal{P}(x, y) = \mathbb{E}\left[l(x, y, f)\right]. \tag{2.3}$$

Naturally, we are interested in the optimal function f^* from $\mathcal{F}$, which minimizes the risk. This ideal estimator is the *Bayes classifier*:

$$f^*(x) = \begin{cases} 1, & \text{if } \mathcal{P}_{Y|X}\left(Y = 1 | X = x\right) \geq \frac{1}{2} \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

The Bayes classifier achieves the infimum of the risk over all possible estimators in $\mathcal{F}$. This optimal risk is called the *Bayes risk*, and it is defined as

$$R^* = \inf_{f \in \mathcal{F}} R(f). \tag{2.5}$$

In practice, neither the Bayes classifier nor the risk can be directly calculated, because the underlying distribution $\mathcal{P}$ is unknown at the time of learning. In this context, the problem of learning has to be formulated as finding a classifier f, from the space of functions $\mathcal{F}$, with risk as close as possible to the Bayes risk.

Even though the actual distribution is unknown, a finite number of observations drawn from $\mathcal{P}$, that is, the training data D, are available for learning. It turns out that these examples can be used to approximate the true risk $R(f)$, by means of *empirical risk* defined as follows:

$$R_{emp}(f) = \frac{1}{n} \sum_{i=1}^{n} l(x_i, y_i, f). \tag{2.6}$$

2.1.2 Risk Minimization

With the hope of learning the underlying distribution from the training set and then *generalizing* to unseen examples, many learning algorithms, such as certain neural network models [24], adopt the strategy of *empirical risk minimization* (ERM). This inductive principle chooses the estimator $\hat{f} \in \mathcal{F}$ that yields the minimum empirical risk

$$\hat{f} = \arg\min_{f \in \mathcal{F}} R_{emp}(f). \tag{2.7}$$

A natural question arises here, namely, to what extent the empirical risk is a good approximation of the true risk. According to the law of large numbers, it is possible to give conditions that ensure that the empirical risk will converge to the true risk, as the number of training examples tends to infinity [21]

$$\lim_{n \to \infty} R_{emp}(f) = R(f). \tag{2.8}$$

However, for an arbitrary and large space of functions $\mathcal{F}$, minimizing the empirical risk on the training set can prove problematic [25]. First, this is usually an *ill-posed* problem because for a given training set, there might be many optimal solutions that minimize the empirical risk. Second, even when a classifier f perfectly predicts the training data, i.e., $R_{emp}(f) = 0$, it almost certainly will not show good *generalization* on unseen data. In cases wherein $R_{emp}(f)$ is minimum but $R(f)$ is large, unwanted *overfitting* takes place. To illustrate this point, consider a classifier g that "remembers" the class labels for all training examples by querying a lookup table. It follows that empirical risk is $R_{emp}(g) = 0$, however, g cannot correctly classify unseen data.

One way to avoid overfitting is by restricting the space of functions $\mathcal{F}$ from which the estimator f is chosen [21]. Without such restriction, the minimization of the empirical risk is not *consistent*. Consistency is determined by the worst case over all estimators that can be implemented. That is, we need a version of the law of large numbers that is uniform over all estimators. If the risk minimization is consistent, the minimum of $R_{emp}(f)$ converges to

$R(f)$ in probability [26].

In SLT, a probabilistic bound can be provided for the difference $|R_{emp}(f)-R(f)|$ in such a way that overfitting can be controlled by minimizing the bound. Interestingly, the bound is independent of the underlying distribution, but it is still assumed that both past and future data are i.i.d. from the same distribution. According to the Vapnik-Chervonenkis (VC) theory [21], tighter bounds depend on both the empirical risk and the *capacity* of the function space, which is a measure of its complexity. In VC theory, the best known capacity concept is the *VC dimension*, defined as the largest number h of points that can be separated for all possible labelings using functions from the space $\mathcal{F}$ [27]. If no h exists, the VC dimension is $+\infty$.

An example of a VC bound is the following: let the VC dimension $h < n$ for a given space $\mathcal{F}$, then for all estimators from that class and a 0-1 loss function, the following bound holds with probability at least $1 - \delta$ [21]:

$$R(f) \leq R_{emp}(f) + \sqrt{\frac{h\left(\log \frac{2n}{h} + 1\right) - \log \frac{\delta}{4}}{n}}. \tag{2.9}$$

The second term on the right hand side of Eq. 2.9 is an increasing function of $\frac{h}{n}$ and δ. That is, overfitting is very likely to occur as the ratio between the capacity and the number of examples in the training set increases. The capacity plays an interesting role, because a very small capacity can reduce the second term but it might not be good enough to learn the training data, yielding a large empirical risk. On the other hand, a very large capacity will reduce the empirical risk but it will increase the second term. This situation is analogous to the bias-variance dilemma from neural networks and statistics [8]. In order to obtain better generalization, the function space must be restricted such that the capacity is as small as possible, given the available training data. Figure 2.1 shows the intuition behind Eq. 2.9.

Unfortunately, bounds introduced by VC theory are very difficult to measure in practice [8], but they can be exploited using the principle of *structural risk minimization* (SRM). The idea here is to define a nested family of function classes (spaces) $\mathcal{F}_1 \subset \cdots \subset \mathcal{F}_k$ with

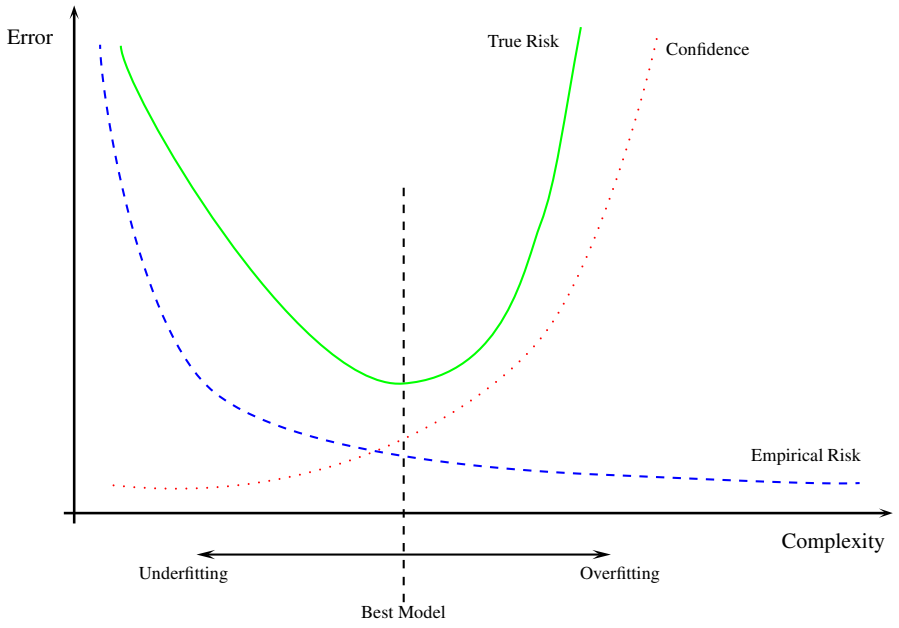


Fig. 2.1: Illustration of Eq. 2.9. Note how the empirical error decreases with higher capacity but the upper bound on the risk (confidence) becomes worse. The best model lies on the best trade-off between complexity and empirical error.

their corresponding VC dimensions satisfying $h_1 \leq \cdots \leq h_k$. Given this sequence of increasingly more complex function spaces, SRM consists of choosing the minimizer of the empirical risk in the function space for which the bound on the structural risk (right hand side of Eq. 2.9) is minimized [25].

2.2 Support Vector Machines

Although bounds as defined previously suffer from practical problems, they offer principled ways to formulate learning algorithms. SVMs, one of the most well-known classifiers, are built on the basis of SRM. To introduce the basic concepts of SVMs, let us assume that the input space is restricted to $\mathcal{X} = \mathbb{R}^d$. Furthermore, under the context of binary classification, i.e., $\mathcal{Y} = \{-1, +1\}$, we assume that the data belong to two classes (see Figure 2.2), that can be *linearly separable* by a hyperplane of the form:

$$f(x) = \langle w, x \rangle + b = 0 \tag{2.10}$$

where the *decision surface* is parametrized by the *weight vector* $w \in \mathbb{R}^d$ and the threshold $b \in \mathbb{R}$, and $\langle w, x \rangle$ denotes the dot product between w and x.

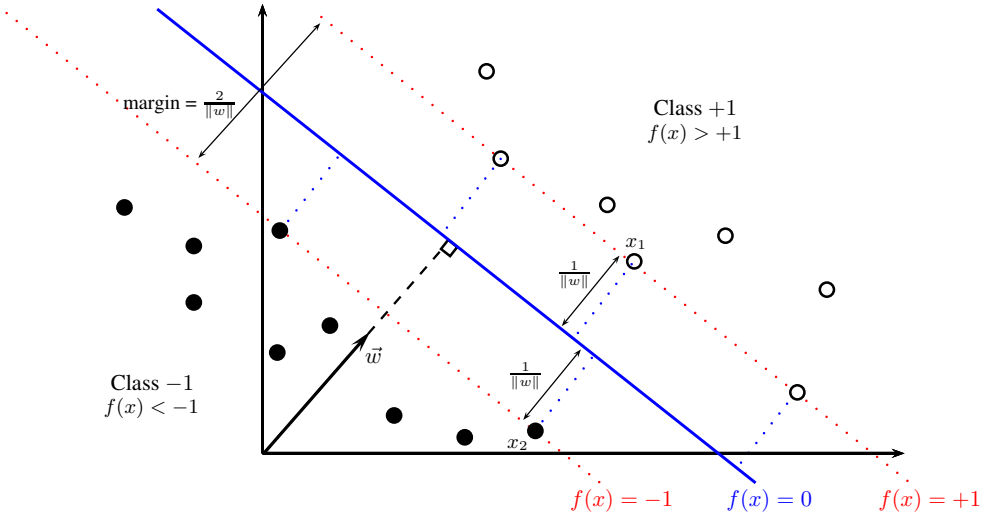


Fig. 2.2: A linear classifier separating two classes. The decision surface (in blue) is a hyperplane defined by $\langle w, x \rangle + b = 0$. The margin is defined by the distance of the closest points (x_1 and x_2).

Consider two points x_1 and x_2 lying on the decision surface, i.e., $f(x_1) = f(x_2) = 0$. Then, using Eq 2.10 we obtain $\langle w, (x_1 - x_2) \rangle = 0$, which reveals that the weight vector w is orthogonal to the decision surface; therefore, it determines the orientation of the hyperplane. Additionally, note that the value $\frac{b}{\|w\|}$ determines the perpendicular distance from the hyperplane to the origin.

All the hyperplanes satisfying $y_i f(x_i) > 0$, for all $i = 1, \ldots, n$, can be considered as decision surfaces. With the purpose of selecting the best hyperplane, it turns out that it is possible to restrict the class of hyperplanes using theoretical insights from VC theory. More specifically, the VC dimension can be bounded in terms of the *margin*. The margin is defined as the minimal distance between an example and the decision surface [8].

A *canonical representation* of the hyperplane is obtained by rescaling w and b such that the closest points to the decision surface satisfy $|\langle w, x_i \rangle + b| = 1$. Now consider two points x_1 and x_2 such that $\langle w, x_1 \rangle + b = +1$ and $\langle w, x_2 \rangle + b = -1$. After projecting both points onto the normal weight vector $\frac{w}{\|w\|}$, the margin is given by the distance between

those projections, i.e., perpendicular to the hyperplane. In consequence, the margin can be expressed in terms of w, because $\frac{w}{\|w\|} \cdot (x_1 - x_2) = \frac{2}{\|w\|}$. These properties are geometrically illustrated in Figure 2.2 for the case $d = 2$.

It is proven in [21] that the larger the margin of a function from $\mathcal{F}$, the smaller its VC dimension. Bear in mind that different functions (hyperplanes) can be defined by simply changing w and b. As the margin of separating hyperplanes expresses the capacity, the SVM algorithm aims at finding the hyperplane parametrization that yields the largest margin for separating all examples in the training set.

2.2.1 Hard Margin

For canonical separating hyperplanes of the form depicted by Eq. 2.10, one can achieve perfect classification of training examples when $y_i(\langle w, x_i \rangle + b) \geq 1$, for $i = 1, \ldots, n$. In other words, when the data are linearly separable, the empirical risk can be kept zero by constraining the parameters w and b. In order to minimize the bound from Eq. 2.9, the complexity term, which increases monotonically with the VC dimension, can be controlled by maximizing the margin. That is, we can minimize the complexity by minimizing $\|w\|$. This is nicely formulated as the following quadratic optimization problem (also known as the *primal* formulation) [8]:

$$\begin{aligned} \underset{w,b}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 \\ \text{subject to} \quad & y_i(\langle w, x_i \rangle + b) \geq 1, \qquad i = 1, \ldots, n. \end{aligned} \tag{2.11}$$

The optimization problem above is convex with linear constraints. We can introduce Lagrange multipliers in such a way that the constraints are replaced by constraints on the multipliers themselves, making the problem easier to handle. Furthermore, in this new formulation, the operations on the examples appear only in the form of dot products. Later, we will generalize the SVM linear model to nonlinear cases by taking advantage of this property.

In the Lagrangian formulation, we introduce multipliers $\alpha_i \geq 0$, $i = 1, \ldots, n$, one for

each of the constraints defined in Eq. 2.11. To form the Lagrangian, the rule is that for constraints of the form $c_i \geq 0$, the constraint equations are multiplied by the α's and subtracted from the objective function; therefore, from Eq. 2.11 we get the following Lagrangian:

$$\begin{aligned} L(w,b,\alpha) &= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\langle w, x_i\rangle + b) - 1\right) \\ &= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i y_i(\langle w, x_i\rangle + b) + \sum_{i=1}^{n} \alpha_i. \end{aligned} \tag{2.12}$$

Once this is a convex quadratic programming problem and those points satisfying the constraints also form a convex set, we can equivalently formulate and solve the *dual* problem [28], wherein the solution is determined by minimizing $L(w,b,\alpha)$ with respect to w, b and maximizing it with respect to α_i. It follows that, the saddle points are given by the conditions:

$$\frac{\partial L(w,b,\alpha)}{\partial b} = 0 \qquad \text{and} \qquad \frac{\partial L(w,b,\alpha)}{\partial w} = 0 \tag{2.13}$$

which after rearrangement of terms translate into:

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \qquad \text{and} \qquad w = \sum_{i=1}^{n} \alpha_i y_i x_i. \tag{2.14}$$

A further expansion of terms in Eq. 2.12 yields

$$L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i y_i \langle w, x_i\rangle - b\sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i \tag{2.15}$$

where the third term on the right-hand side is zero because of the first condition of Eq. 2.14. In addition, we can get rid of w by expanding $\|w\|^2$ as follows:

$$\|w\|^2 = \langle w, w\rangle = \sum_{i=1}^{n} \alpha_i y_i \langle w, x_i\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j\rangle. \tag{2.16}$$

Accordingly, from Eq. 2.15 and 2.16, the dual problem can be simplified and stated as:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{subject to} \quad & \alpha_i \geq 0, \qquad i = 1, \ldots, n \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{2.17}$$

where the solution consists of optimal Lagrange multipliers, denoted by α_i^*. Consequently, the optimum weight vector w^* can be recovered by using the vector α^* in one of the saddle points shown in Eq. 2.14 as follows:

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i. \tag{2.18}$$

In the solution, there are particular cases called *support vectors*. Those are the points lying on one of the hyperplanes $f(x) = 1$ or $f(x) = -1$ for which $\alpha_i^* > 0$, while all training examples that are not support vectors have $\alpha_i^* = 0$.

Once we have the optimum weights, the threshold can be determined by considering that for any support vector x_i, $y_i(\langle w, x_i \rangle + b) = 1$, and thus, $b^* = y_i - \langle w^*, x \rangle$. In fact, instead of calculating b^* for only one support vector, averaging over all support vectors yields a better and more numerically stable solution. With the optimum weights and threshold, the classification rule can be expressed as:

$$f(x) = \operatorname{sgn}\left(\langle w^*, x \rangle + b^*\right) = \operatorname{sgn}\left(\sum_{i=1}^{n} \alpha_i^* y_i \langle x_i, x \rangle + b^*\right). \tag{2.19}$$

In summary, the hard margin formulation of SVMs is applicable when the data are linearly separable. Learning involves solving the optimization problem stated in Eq. 2.17. The solution is known to be *sparse*, in the sense that, for making new predictions we only have to "remember" the support vectors. Note also that in both the dual formulation (Eq. 2.17) and the classification rule (Eq. 2.19), the data appear in the form of dot products. This is a key observation that later will play a role in the definition of SVMs for nonlinear

cases.

2.2.2 Soft Margin

The formulation described in the previous section is applicable when the data are linearly separable, corresponding to an empirical error of zero. Nevertheless, in real world applications, data appear under more complex circumstances and might not be separable by a linear hyperplane. When data from both classes overlap, we need to modify the SVM formulation so as to allow for misclassifications of some training points. The idea is to allow data points to be on the wrong side of the margin; however, they are penalized with an amount proportional to their distance from the margin. To this end, *slack* variables $\xi_i \geq 0$ for $i = 1, \ldots, n$, for each training example are introduced [23]. These variables are defined by $\xi_i = 0$ for points either on the margin or on the correct side of the margin, and $\xi_i = |y_i - f(x_i)|$ for all other cases. Points with $\xi_i > 1$ are misclassified because they lie on the wrong side of the decision surface, and those points for which $0 < \xi_i \leq 1$ lie inside the margin but on the correct side of the decision surface, as illustrated in Figure 2.3.

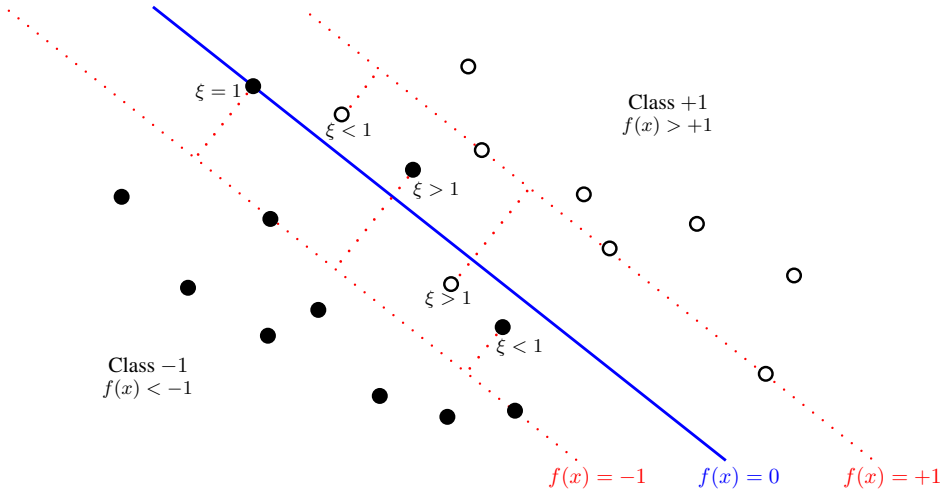


Fig. 2.3: Illustration of the slack variables ξ_i, for $i = 1, \ldots, n$. Note that only the values $\xi_i \neq 0$ are shown, corresponding to points on the wrong side of the margin. All the other points lying either on the margin or on the correct side have $\xi_i = 0$.

The idea of a soft margin in SVMs comes from the relaxation of the hard margin

constraints with the use of slack variables. Hence, the margin has to be maximized in the presence of misclassifications, then the primal problem is formulated in such a way that the VC dimension is small (large margin) while the empirical risk (slack variables) is minimized:

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0, \qquad i = 1, \ldots, n \end{aligned} \tag{2.20}$$

where the *regularization* constant $C > 0$ is used to determine the tradeoff between the margin and the empirical error. Similar to the case of hard margin SVMs, the primal above can be equivalently written as a dual problem:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \qquad i = 1, \ldots, n \\ & \sum_{i=1}^{n}\alpha_i y_i = 0. \end{aligned} \tag{2.21}$$

The dual problem for the soft margin SVM (Eq. 2.21) differs from that of the hard margin (Eq. 2.17) only on the constraints applied to the Lagrange multipliers. In addition, for the soft margin the following observations hold. Data points outside the margin will have $\alpha_i^* = 0$, while points on the margin line have $0 \leq \alpha_i^* \leq C$. Points within the margin have $\alpha_i^* = C$, including misclassified and correctly classified points. Therefore, support vectors are characterized by $0 < \alpha_i^* < C$. The optimum values for the weight vector w^* and threshold b^* can be calculated in the same way as for the hard margin case.

2.3 Kernels

With the introduction of soft margin SVMs, misclassification errors are allowed in order to deal with noisy data. However, in more complex scenarios, the data might not be separable by a hyperplane at all. Thus, the choice of linear functions is, to a certain extent, restricted for learning nonlinearities in the data. We need to generalize the SVM theory

to deal with nonlinear decision surfaces. Opportunely, the use of kernel functions makes possible training linear models, while at the same time, decision functions are nonlinear. In this section, we provide a short description of kernel functions that serve as important background for the development of later chapters. For the corresponding formal proofs and further details, refer to [29].

2.3.1 Kernel Functions and the Kernel Trick

The basic idea is to map the *input space* $\mathcal{X}$ into a potentially richer and higher dimensional *feature space* $\mathcal{H}$. The hope is that nonlinear characteristics of the input space become linear in the enlarged space, therefore, allowing for linear separation with hyperplanes. The nonlinear mapping is denoted by a function $\Phi\colon \mathcal{X} \to \mathcal{H}$ called the *feature map*, illustrated in Figure 2.4.

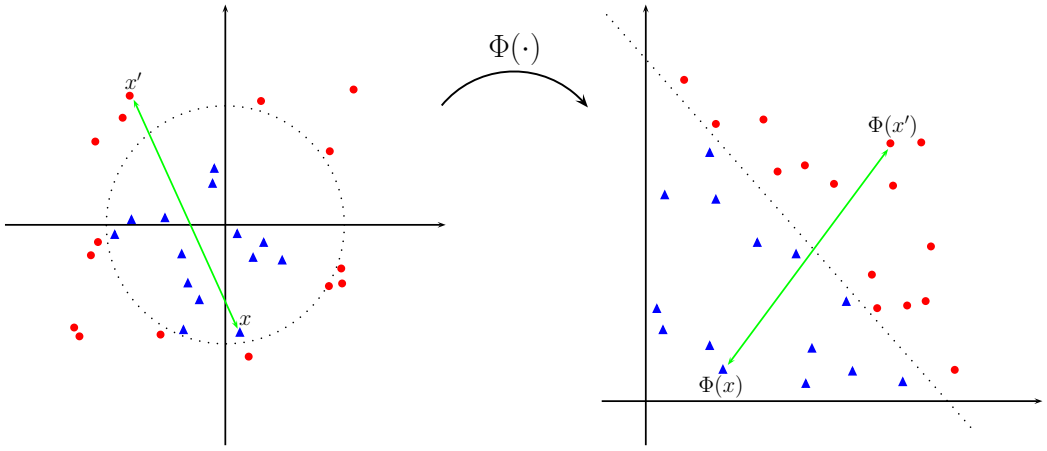


Fig. 2.4: Data points are mapped from an input space $\mathcal{X}$ into a feature space $\mathcal{H}$ by the function Φ. The dashed boundary shows how data points that are not linearly separable in $\mathcal{X}$ become separable in $\mathcal{H}$.

We can make any learning algorithm work in the feature space $\mathcal{H}$ by transforming the input examples into $(\Phi(x_1), y_1), \ldots, (\Phi(x_n), y_n) \in \mathcal{H} \times \mathcal{Y}$. Going into a higher dimensional space may appear to suffer from the *curse of dimensionality* problem, therefore requiring many more examples. However, according to SLT, learning in $\mathcal{H}$ can be simpler if one uses a low complexity class of decision functions, for example, classifiers based on hyperplanes [8].

While it is true that for certain applications we can define and apply an appropriate function $\Phi(\cdot)$, the explicit mapping to a higher dimensional space $\mathcal{H}$ can be intractable due to the space required and the computational cost involved in the transformation of all examples. It turns out that for certain spaces, there is an attractive way to implicitly calculate dot products in $\mathcal{H}$ without even knowing the function $\Phi(\cdot)$:

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle \tag{2.22}$$

where $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is known as the *kernel* function. Such a kernel can be intuitively thought of as a similarity measure for any pair of objects from $\mathcal{X}$.

Learning algorithms can take advantage of kernel functions by implementing the *kernel trick*. Any algorithm that exclusively uses dot products to interact with input examples from $\mathcal{X}$ can use a kernel function instead to solve the problem in the feature space $\mathcal{H}$. There is no need to compute the mapping $\Phi(\cdot)$ explicitly. For example, the soft SVM from Eq. 2.21 can be extended to a nonlinear space by replacing the dot product $\langle x_i, x_j \rangle$ with a kernel function $k(x_i, x_j)$.

2.3.2 Positive Definite Kernels

In order to make possible similarities in the input space corresponding to dot products in a feature space, kernel functions must be *valid*. A valid kernel function satisfies *symmetry*, i.e., $k(x_i, x_j) = k(x_j, x_i)$, and *positive definiteness* in the following sense:

Definition 1 (positive definite kernel) *A symmetric function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite* [1] *kernel if, for any $n \in \mathbb{N}, x_1, \ldots, x_n \in \mathcal{X}$, and $c_1, \ldots, c_n \in \mathbb{R}$*

$$\sum_{i=1,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0. \tag{2.23}$$

[1] In mathematics, functions for which this sum is strictly positive when $c_i \neq 0$ and $c_j \neq 0$ are called positive definite functions, whereas functions for which this sum is only non-negative are called positive semidefinite functions. For brevity, we will use the term positive definite indifferently for both cases.

Definition 2 (kernel matrix) *Given a positive definite kernel function k and a set of examples $x_1, \ldots, x_n \in \mathcal{X}$, the $n \times n$ matrix K such that $K(i,j) = k(x_i, x_j)$ for $i, j = 1, \ldots, n$ is called the kernel matrix (or gram matrix).*

From the definitions above, it follows that if k is a positive definite kernel, we can construct a *Hilbert space* $\mathcal{H}$ in which k is a dot product. It is shown in [28] that every kernel function is associated with a Reproducing Kernel Hilbert Space (RKHS) and that every RKHS is associated with a kernel function. Furthermore, Eq. 2.23 is considered essential for practitioners. One of the most powerful kernel methods for classification, the SVM, has the advantage of finding a unique solution. This is only possible when the optimization problem is convex. It turns out that the SVM problem is convex whenever the used kernel matrix satisfies the definitions above.

2.3.3 Properties of Kernels and Examples

Positive definite kernel functions have attractive properties that can be used to create new kernel functions, by combining existing ones. Assuming that $\mathcal{X}$ is an input space and k_1 and k_2 are arbitrary positive definite kernels defined over $\mathcal{X} \times \mathcal{X}$, the following are also positive definite kernels: $\alpha \cdot k_1$, for $\alpha > 0$; $k_1 + k_2$; $k_1 \cdot k_2$; and $\exp(k_1)$.

The distance between two points in the feature space can also be calculated using the kernel trick. Consider two objects $x_i, x_j \in \mathcal{X}$ such that, their respective mappings are $\Phi(x_i)$ and $\Phi(x_j)$ in $\mathcal{H}$. The distance d_{ij} in $\mathcal{H}$ can be computed as:

$$\begin{aligned} d_{ij} &= \|\Phi(x_i) - \Phi(x_j)\| \\ &= (\langle \Phi(x_i), \Phi(x_i) \rangle + \langle \Phi(x_j), \Phi(x_j) \rangle - 2\langle \Phi(x_i), \Phi(x_j) \rangle)^{\frac{1}{2}} \\ &= \sqrt{k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j)} \end{aligned} \tag{2.24}$$

where k is a positive definite kernel function.

It is often a good practice to normalize or scale the training data before applying kernel functions. This is trivial when the data come as a set of vectors, because we can apply the transformation to each column. In a more general case, we only have the kernel

matrix. Sometimes the kernel matrix is created from objects from a complex input space, not necessarily a Euclidean space. In these cases, it is considered effective to normalize the kernel matrix such that the diagonal elements are all equal to 1, according to:

$$K(i,j) = \frac{K(i,j)}{\sqrt{K(i,i) \cdot K(j,j)}} \qquad \text{for all } i,j = 1,\ldots,n \tag{2.25}$$

where $K(i,j)$ denotes the (i,j)th element of the kernel matrix K [30].

A few instances of kernels for vectorial data have been very popular among SVM practitioners, and they are implemented by default in widely used software packages for SVMs, such as *LibSVM*[2] and *SVMlight*.[3] Table 2.1 shows a list of well-known positive definite kernels.

Table 2.1: List of known positive definite kernels for vectorial data.

Kernel	Definition	Parameters
Linear	$\langle x_i, x_j \rangle$	
Polynomial	$(\langle x_i, x_j \rangle + c)^p$	$p \in \mathbb{N}, c \geq 0$
Gaussian	$\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$	$\sigma > 0$

In summary, there are two major benefits in the use of kernels. First, it allows the definition of similarities between arbitrary objects. Once there are no restrictions on the input space $\mathcal{X}$, kernels can be designed for any kind of data, as long as they are valid kernels. For example, several kernels have been proposed to work with complex objects, such as strings, sequences, trees, or graphs (illustrated in Figure 2.5). Second, valid kernel functions can be embedded in any learning task, that is, clustering, classification, regression, or feature extraction, provided that the respective algorithms are based on dot product calculations. For instance, the family of kernel methods includes learning algorithms where observations in the training set only come into the algorithm via dot product calculations with each other.

[2] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`
[3] `http://svmlight.joachims.org/`

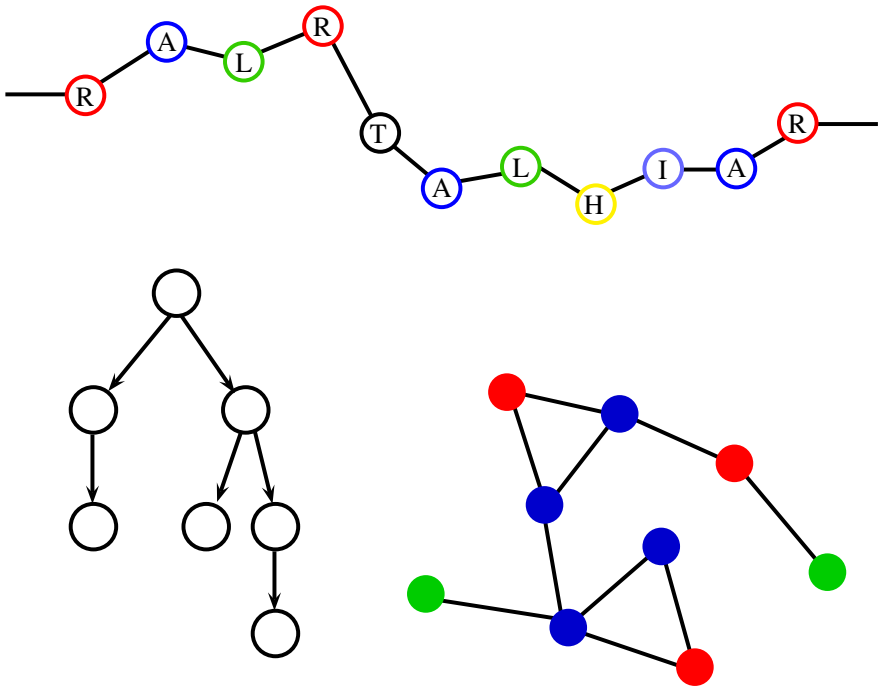


Fig. 2.5: Instances of structured objects, such as sequences, trees and, undirected graphs. One of the major advantages of kernel functions is that they can be defined to measure the similarity between structured objects.

Chapter 3

Kernels for Graphs

In this chapter, several kernels for graphs are reviewed. Graph kernels are a central topic in this dissertation, since the computational biology problems addressed in later chapters are based on graph models of proteins. One can think of graph kernels as similarity functions designed for graph structures. Such similarities must be valid kernels in order to be employed in kernel-based learning tasks as classification or clustering. The design of graph kernels is based on a rich set of fundamentals from graph theory. The challenge is to define similarity measures capable of capturing the structural commonalities between pairs of graphs.

Initially, a brief introduction to graph theory [31] is presented with the purpose of defining terminology and notation for the remaining sections. Next, a brief review of existing algorithms for graph matching is presented (for more details refer to [32]). Then, we present a concise analysis of the major kernel functions for graphs existing in the literature. When possible, we make their connection with graph matching algorithms, as well as highlighting their advantages and shortcomings.

3.1 Introduction to Graph Theory

Formally, a *graph* G is defined as a pair $\langle V, E \rangle$, where V denotes the nonempty set of *vertices* (*nodes*), and $E \subseteq V \times V$ denotes the set of *edges*. An edge $e \in E$ connects a pair of vertices $u, v \in V$, and it is denoted as (u, v). The *order* of a graph G, denoted by $|G|$, is defined as the number of vertices in the graph. A graph of order 1 is called *trivial*. If the vertices or edges of a graph are assigned labels we obtain a *labeled* graph, sometimes referred to as an *attributed* graph. When graphs are used to model real world artifacts, e.g., protein structures, the vertices are used to represent entities, and the edges are the

relationships between pairs of entities.

Vertices $u, v \in V$ are *adjacent* if there exists $e = (u, v) \in E$, where edge e is called *incident* to the vertices u and v. When two vertices are adjacent, they are called *neighbors*. The *degree* of a vertex v is defined as the number of incident edges to v. We define the *adjacency matrix* of an unlabeled graph $G = \langle V, E \rangle$ as $A_{n \times n} = [a_{ij}]$ where $a_{ij} = 1$ if (v_i, v_j) is an edge of G and 0 otherwise.

When the edges in E are ordered pairs of vertices, the graph is called *directed*. Any directed edge connects a *source* vertex to a *target* vertex. Graphs are *undirected* when the edges in E do not have a particular order or direction. A graph is called *simple* if there is no edge connecting a vertex to itself (*loop*) and there are no *multiple edges* connecting the same pair of vertices. A *multigraph* is a graph that contains multiple edges. A graph is *complete* if all of their vertices are connected to each other. The complete graph with n vertices is denoted by K_n. Graphs can be represented pictorially, as in Figure 3.1.

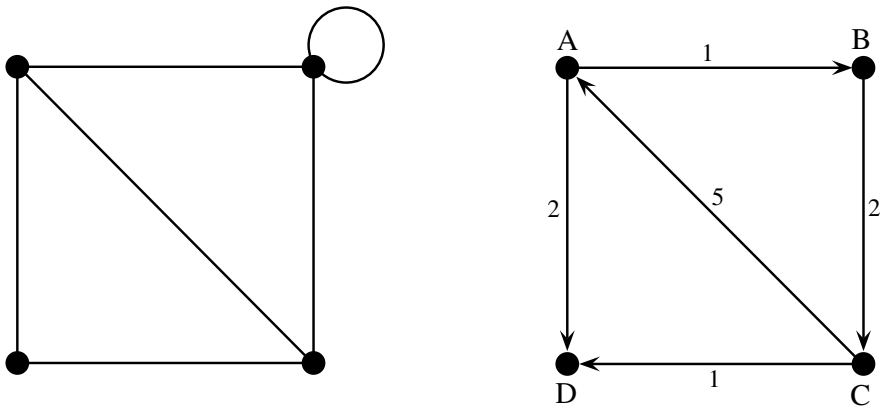


Fig. 3.1: Illustration of a graphical representation of graphs. The graph on the left is an undirected graph containing one loop. The graph on the right is a labeled directed and simple graph.

Let $G \cup G' = \langle V \cup V', E \cup E' \rangle$ and $G \cap G' = \langle V \cap V', E \cap E' \rangle$. If $G \cap G' = \emptyset$, then G and G' are *disjoint* graphs. If $V' \subseteq V$ and $E' \subseteq E$, then G' is a *subgraph* of G (also written as $G' \subseteq G$), and G is a *supergraph* of G'. A complete subgraph from G is referred to as a *clique*. If $G' \subseteq G$ and E' contains all the edges $e = (u, v)$ such that $e \in E$ with $u, v \in V'$, we say that G' is an *induced subgraph* of G.

A *walk* w in a graph G is a nonempty sequence of vertices $v_1, \ldots, v_k$ connected by edges $e_1, e_2, \ldots, e_{k-1}$ such that $e_i = (v_i, v_{i+1})$ for all $1 \leq i < k$. The *length* of a walk is the number of edges in the sequence. If the vertices in the sequence are all distinct, w is called a *path* p in G. If $v_1 = v_k$ in the path p, p is called a *cycle* in G. We can also say that a graph G is *connected* if for every pair of distinct vertices in G, there exists a path connecting both vertices.

Let $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$ be two graphs. The graphs G and G' are *isomorphic*, denoted by $G \simeq G'$, if there exists a bijection $f\colon V \to V'$ with $(u, v) \in E \Leftrightarrow (f(u), f(v)) \in E'$ for all $u, v \in V$. The map f is called an *isomorphism*. This bijection expresses the general notion of isomorphism being a structure preserving function. However, further restrictions may be imposed so that additional elements are preserved. For example, vertex labels commonly taken from a discrete alphabet. The graph isomorphism relation satisfies the conditions of reflexivity, symmetry, and transitivity [33].

3.2 Graph Matching

Algorithms for graph matching can roughly be classified into two broad categories: methods that aim to determine whether two graphs or subgraphs are identical by means of an *exact matching*, and methods that are error tolerant by means of *inexact matching* [33]. Note that while exact matching for strings, sequences, or vectors are trivial tasks, the same task applied to pairs of graphs is much more complex. Checking if two graphs are identical involves checking if the graphs are isomorphic, or in some cases, if the graphs are identical in terms of topology and labels.

3.2.1 Algorithms Based on Graph Isomorphism

In general, checking isomorphism between graphs demands an exponential computational cost with respect to the number of vertices of the involved graphs. A weaker form of matching is a particular case of isomorphism called *subgraph isomorphism*, which exists between two graphs G and H, if the larger graph H can be turned into a graph that is isomorphic to G by removing some edges and nodes. From this definition it follows that

G is contained in H. Note that matching graphs by means of graph and subgraph isomorphism implies a binary decision, that says whether the graphs are identical or not. This is a shortcoming when the goal is to infer graph similarities. Imagine two graphs that are not isomorphic but present several identical vertices and edges. In such a case, the existing degree of similarity between the graphs is not taken into account by methods based on graph and subgraph isomorphism.

In order to avoid this situation, the *maximum common subgraph* can be calculated. Let G and H be graphs, a graph F is called a common subgraph of G and H if F is isomorphic to G and H. Then, the maximum common subgraph is the common subgraph with the maximum order. The solution to this problem is not unique, in the sense that there may be several common subgraphs with the same maximum order.

The similarity (or distance) between graphs can be defined in terms of the maximum common subgraph. Intuitively, the larger the common subgraph, the higher the similarity. Although this might work for graphs with discrete labels or unlabeled graphs, the presence of continuous values in labels makes computing similarity between graphs practically impossible. Imagine two graphs with identical structures consisting of very similar but not identical continuous labels. In this case, the resultant maximum common subgraph is an empty graph, because its calculation relies on isomorphism checks.

All the matching approaches mentioned above are NP-complete, with the exception of graph isomorphism, for which there is no proof that it belongs to the class NP-complete. Although polynomial time algorithms exist for particular and constrained instances of graphs, no polynomial time algorithms exist for the general case. Thus, exact matching or at least matching among subparts, has prohibitive time complexity for large graphs in the worst case [32].

3.2.2 Graph Edit Distance

Given the shortcomings of exact graph matching methods, their applicability to real world problems is restricted. To overcome this, several inexact matching methods have been proposed. The idea of inexact matching is that the algorithms evaluate similarities between

graphs by relaxing the constraints that define the matching. A very common approach along these lines is the *graph edit distance*, roughly defined as the cost of the minimum amount of edit operations that is required to transform one graph into the other. A standard set of edit operations include insertions, deletions, and substitutions of both vertices and edges. Graph comparison methods based on edit distance are convenient because they are expressive and they can be applied to arbitrary graphs, where structural errors and continuous labels are allowed. However, they are hard to parametrize and have a high cost associated with time and space complexity. Therefore, these methods are only applicable to relatively small graphs. For a detailed survey of graph edit distance methods, refer to [34].

3.2.3 Graph Embedding

Another approach for graph comparison is *graph embedding* in vector spaces. The goal is to find feature vector representations in a real vector space $\mathbb{R}^n$ for graphs from some graph domain $\mathcal{G}$. These feature vectors are also referred to as *topological descriptors*. Formally, embedding is denoted by a function $\psi\colon \mathcal{G} \rightarrow \mathbb{R}^n$. The motivation for graph embedding methods is that the whole arsenal of pattern recognition tools developed for feature vectors becomes automatically available for the domain of graphs. A popular class of graph embedding methods is based on spectral graph theory [35,36]. Here, graph features are characterized using the spectral decomposition of the Laplacian matrix [36]. A major limitation of methods based on spectral graph theory is that the decomposition is sensitive to structural errors, such as missing or spurious vertices. Furthermore, spectral methods are only applicable to unlabeled graphs or labeled graphs with restricted label alphabets [37].

Recently, a new class of graph embedding methods tolerant to structural errors that allows graphs with arbitrary labels on nodes and edges has been proposed [38]. The basic idea of this approach is the calculation of distances from an input graph G to a number of particular graphs selected from the training set called *prototype graphs*. The resulting distances are expressed as a vectorial signature of G. Unfortunately, despite the advantages offered by machine learning tools for vectorial data, methods based on graph embedding still suffer from the high cost in runtime complexity. Computation of topological descriptors may

require exponential runtime. In fact, graph embedding using prototype graphs resorts to the calculation of graph edit distances between G and the prototypes. Hence, the calculation of optimal solutions is prohibitive for large graphs because the edit distance of graphs is exponential in the number of nodes of the involved graphs. Therefore, the embedding is carried out using approximation algorithms with polynomial time.

3.3 Review of Graph Kernels

Kernel functions for graphs can make the connection between structural data and the powerful set of tools and learning algorithms called kernel methods. Intuitively, a graph kernel is a measure of similarity between two input graphs satisfying the conditions of symmetry and positive definiteness. The formal definition of a graph kernel is given below.

Definition 3 (graph kernel) *Let $\mathcal{G}$ be the domain of graphs. The function $k\colon \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is called a graph kernel if there exists a Hilbert space $\mathcal{H}$ and a mapping $\Phi\colon \mathcal{G} \to \mathcal{H}$ such that:*

$$k(G, G') = \langle \Phi(G), \Phi(G') \rangle \qquad \textit{for all } G, G' \in \mathcal{G}. \tag{3.1}$$

3.3.1 R-Convolution Kernels

Many of the existing kernels for graphs are based on the seminal idea of R-convolution kernels [39], which provides a general framework for dealing with discrete compound objects. In convolution kernels, complex objects are decomposed into smaller parts, for which a simpler similarity measure can be defined and computed more efficiently. Given the similarities between the smaller parts, a convolution operation can be used to define a kernel function between a pair of complex objects. Assume that a sample $x \in \mathcal{X}$ can be decomposed into parts $\vec{x} = x_1, \ldots, x_d \in \mathcal{X}_1, \ldots, \mathcal{X}_d$, for example, the decomposition of graphs into subgraphs. We define the relation R, where $R(\vec{x}, x)$ is true whenever $\vec{x}$ is a valid decomposition of x and false otherwise. Now consider the inverse $R^{-1} = \{\vec{x} | R(\vec{x}, x) = true\}$, the

set of all valid decompositions of an object. The R-convolution of the kernels $k_1, \ldots, k_d$ denoted $k_1 \star k_2 \star \cdots \star k_d(x, x')$ with $k_i \colon \mathcal{X}_i \times \mathcal{X}_i \to \mathbb{R}$ is defined as:

$$k(x, x') = k_1 \star k_2 \star \cdots \star k_d(x, x') = \sum_{\substack{\vec{x} \in R^{-1}(x) \\ \vec{x'} \in R^{-1}(x')}} \prod_{i=1}^{d} k_i(x_i, x_i') \tag{3.2}$$

where $k(x, x')$ is a valid kernel, provided that all the individual $k_1, \ldots, k_d$ are valid kernels and R is a finite relation [39]. This deliberately vague formulation leads to a framework wherein many different kernels can be defined by changing the decomposition. The application of R-convolution kernels to graphs involves the decomposition of graphs into smaller substructures computable in polynomial time. Several instances of graph kernels based on this framework have been proposed, for which some of the most frequently used substructures are random walks, subtrees, cycles, and shortest paths [40].

3.3.2 Random Walk Kernels

The basic idea of random walk kernels is to count the number of matching walks in both graphs. In [41], it is shown that the number of matching walks in two graphs $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$ can be calculated by means of the *direct product graph* $G_\times = G \times G'$, which identifies the compatible vertices and edges between the two input graphs. The sets of vertices and edges for $G_\times$ are respectively defined as:

$$\begin{aligned} V_\times &= \left\{(u, v) \colon u \in V \wedge v \in V' \wedge \mathit{label}(u) = \mathit{label}(v)\right\} \\ E_\times &= \left\{((u, v), (x, y)) : (u, x) \in E \wedge (v, y) \in E' \wedge \mathit{label}(u, x) = \mathit{label}(v, y)\right\}. \end{aligned} \tag{3.3}$$

For any two input graphs G and G', let $A_\times$ denote the adjacency matrix of their direct product graph $G_\times$, if the limit of the matrix power series exists. The product graph kernel is defined as:

$$k_\times(G, G') = \sum_{i,j=1}^{|V_\times|} \left[\sum_{k=0}^{\infty} \lambda_k A_\times^k\right]_{ij} \tag{3.4}$$

where λ is a sequence of weights $\lambda_0, \lambda_1, \ldots$ such that $\lambda_i \in \mathbb{R}$ and $\lambda_i \geq 0$. To compute this

graph kernel, it is necessary to compute the above matrix power series. According to [41], the limit of $k_\times$ can be computed efficiently for two choices of λ.

A geometric series $\sum_i \gamma^i$ is known to converge if and only if $|\gamma| < 1$, where the limit is given by $\lim_{k\to\infty} \sum_{i=0}^{k} \gamma^i = \frac{1}{1-\gamma}$. The geometric series of a matrix can be defined similarly, leading to the geometric random walk kernel as follows:

$$k_{grw}(G, G') = \sum_{i,j=1}^{|V_\times|} \left[\sum_{k=0}^{\infty} \lambda^k A_\times^k\right]_{ij} = \sum_{i,j=1}^{|V_\times|} \left[(I - \lambda A_\times)^{-1}\right]_{ij} \tag{3.5}$$

for $\lambda < \frac{1}{a}$, where a is no less than the maximum degree of a vertex in the direct product graph. As matrix inversion is roughly of cubic time complexity, its application to $(I - \lambda A_\times)$ of size $n^2 \times n^2$ leads to an overall complexity of $O(n^6)$.

Analogously, using exponential series and setting $\lambda_k = \frac{\beta^k}{k!}$, we obtain the exponential random walk kernel as follows:

$$k_{erw}(G, G') = \sum_{i,j=1}^{|V_\times|} \left[\sum_{k=0}^{\infty} \frac{(\beta A_\times)^k}{k!}\right]_{ij} = \sum_{i,j=1}^{|V_\times|} \left[e^{\beta A_\times}\right]_{ij} \tag{3.6}$$

which can be calculated using matrix diagonalization, a matrix eigenvalue problem. Once the matrix $A_\times$ is diagonalized, computing the exponential can be done in linear time [41]. Diagonalization of a matrix is roughly cubic time; thus, the overall complexity for this kernel is again $O(n^6)$.

Once the random walk kernel defined above is constrained to discrete attributes, based on ideas from [42], this kernel is redefined in [43]. The idea originally was to handle continuous labels, by allowing similarities between walks that are not identically labeled. In the modified version, vertex and edge labels along the walks are compared with kernel functions.

Another limitation of this kernel is that the direct product graph might result in $n^2 \times n^2$ matrices, requiring significant time and memory resources. However, in [40], the same problem is stated in terms of Kronecker products that can be exploited in order to reduce the runtime complexity to $O(n^3)$.

Besides the runtime limitations, these graph kernels are subject to *tottering* [44], that

is, the generation of high similarity scores by the presence of redundant small identical substructures. As walks allow for repetitions of vertices, visiting iteratively the same vertices can lead to artificially large contributions to the kernel value. Alternative extensions are proposed in [44] to improve both efficiency and expressiveness. First, vertices are relabeled in order to insert information about the environment of each vertex. Therefore, computation time is reduced because the number of identical labeled paths decreases. Second, the random walk model from [42] is modified in order to avoid tottering, by preventing the walk from coming back to a vertex that was just visited.

3.3.3 Cycle and Subtree Pattern Kernels

A graph kernel based on the decomposition of graphs into cycles and tree patterns is presented in [45]. For each graph G, the set of cyclic patterns, denoted by $\mathcal{C}(G)$, and the set of tree patterns, denoted by $\mathcal{T}(G)$, are calculated. Both sets are induced by the set of simple cycles and bridges of the graph respectively, wherein bridges are defined as the edges not belonging to simple cycles. Simple cycles are cycles with no repeated vertices or edges, aside from the necessary repetition of the start and end vertex. The cycle pattern kernel is then defined as the number of common patterns occurring in both graphs, for which the intersection kernel $k_\cap$ is used as follows:

$$\begin{aligned} k_{cycle}(G, G') &= k_\cap(\mathcal{C}(G) \cup \mathcal{T}(G), \mathcal{C}(G') \cup \mathcal{C}(G')) \\ &= |\mathcal{C}(G) \cap \mathcal{C}(G')| + |\mathcal{T}(G) \cap \mathcal{T}(G')|. \end{aligned} \tag{3.7}$$

The problem of computing cyclic pattern kernels is NP-hard. Since required computations are intractable for the general case, the approach presented in [45] is limited to well-behaved graph databases, wherein graphs are constrained to have a small upper bound on the number of simple cycles. Thus, this function is only applicable to a restricted set of databases.

An alternative formulation based on the number of common subtree patterns in two graphs is given in [46]. Given two input graphs $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$, this subtree

pattern kernel iteratively compares the neighborhoods of all pairs of nodes $u \in V$ and $v \in V'$. The subtree patterns considered may include repeated vertices or edges. Formally, the subtree pattern kernel is defined as:

$$k_{tree}(G, G') = \sum_{u \in V} \sum_{v \in V'} k(u, v, h) \tag{3.8}$$

where $k(u, v, h)$ is the weighted count of pairs of identical subtree patterns with a height less than or equal to h, with the first subtree rooted at $u \in V$ and the second one rooted at $v \in V'$. If $h = 0$ and $label(u) = label(v)$ then $k(u, v, h) = 1$. If $h = 0$ and $label(u) \neq label(v)$, we have $k(u, v, h) = 0$. For $h > 0$, $k(u, v, h)$ can be computed as follows:

$$k(u, v, h) = \lambda_u \lambda_v \sum_{R \in M_{u,v}} \prod_{(u', v') \in R} k(u', v', h - 1) \tag{3.9}$$

where λ_u and λ_v are positive values smaller than 1 to cause higher trees to have a smaller weight in the overall sum, and $M_{u,v}$ is the set of all exact matchings from the set of neighbors of u to the set of neighbors of v.

Although the subtree pattern kernel is based on more expressive substructures, it is still subject to tottering because the same vertices may be visited more than once. Moreover, the complexity of this kernel is $O(n^2 h 4^d)$ for which d denotes an upper bound for the degree of vertices [47]. This kernel is more computationally expensive than kernels based on walks. Motivated by chemical applications, the subtree pattern kernel above is revisited and extended in [48]. The extensions are defined to avoid tottering and to control the complexity of the subtrees. However, the complexity is still exponential in d.

3.3.4 Shortest Path Graph Kernels

Graph kernels based on shortest paths are proposed in [10]. Roughly speaking, this kernel counts the number of shortest paths of the same length having similar start and end vertex labels in two input graphs. One of the motivations for this kernel is the presence of tottering in graph kernels using walks. Unlike walks, vertices are not repeated in paths,

thus, tottering can be avoided by restricting graph kernels to paths. Following the R-convolution kernel, one can easily derive an all-paths kernel. However, the calculation of such a kernel is NP-hard, because by knowing all paths one could determine whether a graph has a Hamiltonian path or not, a problem known to be NP-complete [10]. In a similar way, one can prove that determining all longest paths in a graph is NP-hard.

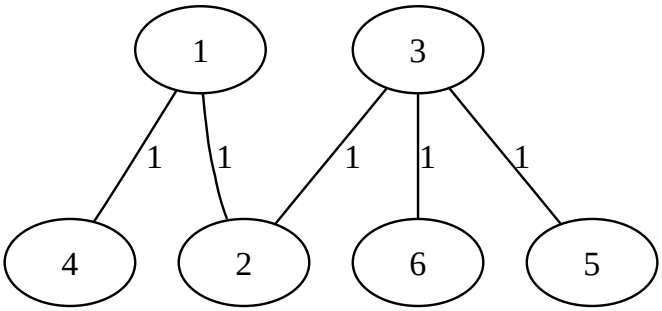
It turns out that the problem of determining all shortest distances in a graph is solvable in polynomial time. The Floyd-Warshall algorithm [49] depicted in Algorithm 3.1, for example, allows the calculation of shortest distances for all pairs of nodes in $O(n^3)$ time, where n denotes the number of vertices. This algorithm allows graphs with negative edge labels, but not containing any negative cycles, which happen when all edge labels in the cycle sum to a negative value. In order to define a kernel that counts shortest paths of similar distances, the original graphs must be transformed into shortest path graphs. This step is a preprocessing requirement before calculating the shortest path graph kernel.

Algorithm 3.1 The Floyd-Warshall Algorithm.

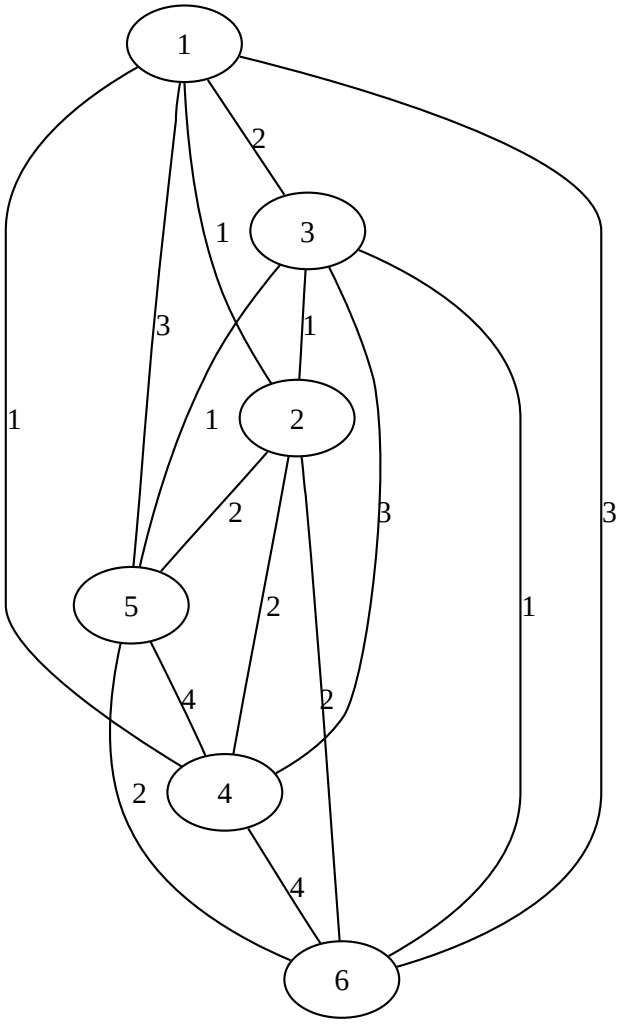
Input: V : set of vertices from G
Output: M : pairwise shortest distance matrix

```
1: M[i][j] ← 0, for every i == j
2: M[i][j] ← 1, for every edge (v_i, v_j) ∈ G
3: M[i][j] ← ∞, for pairs (v_i, v_j) ∉ G
4: n ← length(V)
5: for k ← 1 to n do
6:    for i ← 1 to n do
7:       for j ← 1 to n do
8:          M[i][j] ← min(M[i][j], M[i][k] + M[k][j])
9:       end for
10:   end for
11: end for
```

Given a graph $G = \langle V, E \rangle$, a shortest path graph is a graph $G_{sp} = \langle V', E' \rangle$, where $V' = V$ and $E' = \{e'_1, \ldots, e'_m\}$ such that $e'_i = (u'_i, v'_i)$ if the corresponding vertices u_i and v_i are connected by a path in G. The edges in the shortest path graph are labeled with the shortest distance between the two nodes in the original graph. Figure 3.2 illustrates the transformation of a labeled graph into a shortest path graph.



(a) Original Graph



(b) Shortest Path Graph

Fig. 3.2: Illustration of the transformation of a labeled graph into a shortest path graph. Note that the set of vertices is the same in both graphs. Every edge connecting a pair of vertices in the shortest path graph (3.2(b)) is labeled with the length of the shortest path between such vertices in the original graph (3.2(a)).

A shortest path graph kernel for two shortest path graphs $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$ is defined as:

$$k_{sp}(G, G') = \sum_{e \in E} \sum_{e' \in E'} k_{walk}(e, e') \tag{3.10}$$

where k_{walk} is a positive definite kernel for comparing two edge walks of length 1.

The edge walk kernel k_{walk} is the product of kernels on nodes and edges along the walk. Since the length of the walk is 1, k_{walk} can be calculated in terms of the start vertex, the end vertex, and the edge connecting both. Let e be the edge connecting vertices u and v, and e' be the edge connecting nodes u' and v'. The edge walk kernel is defined as follows:

$$k_{walk}(e, e') = k_{node}(u, u') \cdot k_{edge}(e, e') \cdot k_{node}(v, v') \tag{3.11}$$

where k_{node} is a valid kernel function for comparing two vertices, and k_{edge} is a valid kernel function for comparing two edges. The positive definiteness of the kernel in Eq. 3.10 follows from its definition as a R-convolution kernel.

This kernel is attractive because it retains expressivity while avoiding tottering. Moreover, it can be applied to all graphs on which Floyd-Warshall can be performed, as well as the fact that it allows for continuous labels in vertices and edges. The runtime complexity of this kernel is $O(n^4)$, because the Floyd-Warshall transformation can be done in $O(n^3)$ and the kernel calculation requires a pairwise comparison on the number of edges of the shortest path graphs. The latter takes $O(n^2 * n^2)$, because in the worst case the shortest graph is complete, having n vertices and $\frac{n(n-1)}{2}$ edges. While $O(n^4)$ is an obvious improvement over other graph kernels, it is still expensive for large graphs or datasets. On the other hand, a limitation of this kernel may arise in applications wherein longest paths contain decisive information.

3.4 Other Kernels for Graphs

In addition to the kernels described in previous sections, other approaches have been proposed. The optimal assignment kernel [50] defined for chemical compounds is based on

the idea of computing the optimal assignment from the atoms of one molecule to the atoms of the other. In a general setting, this kernel assigns the vertices from one graph to the vertices of another such that the total similarity between the vertices is maximized. While finding the optimal assignment of substructures sounds more appealing than the all-pairs comparison of R-convolution kernels, the optimal assignment kernel is not always positive definite [40], limiting its use in kernel-based learning methods. Efforts to employ graph edit distances in graph kernels have been presented in [33]. Unfortunately, edit distance does not define a metric, and kernels are not guaranteed to be positive definite [51].

Given that most graph kernels do not scale to large graphs having hundreds or thousands of vertices, a kernel based on counting *graphlets* (small subgraphs of three to five vertices) has been proposed in [52]. Since enumerating all graphlets is unfeasible in practice, two efficient computation schemes are used, which are based respectively on graphlet sampling and the limitation of bounded degree graphs. Let d denote the maximum degree of a graph. All the connected graphlets of size $k = \{3, 4, 5\}$ in a bounded graph G can be enumerated in $O(nd^{k-1})$ time, where n is the number of vertices of G. Although this function can scale to large graphs, it can only be applied to unlabeled graphs. Along the same lines, the Weisfeiler-Lehman test of isomorphism from graph theory is used to calculate fast kernels based on subtrees [52]. These kernels can nicely handle large graphs with discrete labels in $O(mh)$ time, where m is the number of edges and h denotes the height of the subtrees.

3.5 Final Remarks

This chapter presents a review of existing graph kernels. The main goal of such functions is to calculate similarities between pairs of input graphs while being positive definite. Given the complexity inherent in structured data in general, the design of similarity functions that are valid kernels is limited by conflicting quality requirements.

Acceptable graph kernels have to fulfill certain basic conditions. They are required to be expressive, in the sense that they measure similarity by really taking into account the topology and labels on vertices and edges. Meanwhile, a higher expressiveness is frequently

associated with a higher computational cost. For practical purposes, graph kernels are required to have a low computational cost, as well as scaling to large graphs. Due to these conflicting criteria, some conditions are relaxed, yielding more specific graph kernels only appropriate to applications framed under a restricted set of settings. From the graph kernels reviewed in this chapter, we can conclude that a general limitation is given by the high computational cost. In cases where the complexity is acceptable, as in graphlets or fast subtree kernels, such kernels do not allow the use of continuous labels, preventing their use in a wide range of applications.

In later chapters, we present computational biology applications wherein proteins are modeled as graphs. An essential characteristic of such models is the presence of continuous labels at vertices and/or edges. Therefore, we are interested in graph kernels that are not restricted to special classes of graphs. Evidently, an additional criterion is the computational cost. Fast subtree kernels exhibit a very attractive running time. Unfortunately, they are ineligible because according to their formulation, continuous labels are not allowed at vertices. Other kernels are ineligible for the same reason, or because their computational cost is prohibitive. However, the class of shortest path graph kernels is highly appropriate because they retain expressiveness at a relatively low computational cost, when compared with others. Moreover, results from benchmarks presented in [47] show that shortest path graph kernels are competitive, in terms of accuracy and runtime, when compared with several other graph kernels. Experiments were performed on datasets with approximately $4,100$ examples, wherein graphs have a maximum order $|G| = 111$.[1]

The next two chapters discuss specific computational biology problems involving proteins that can be modeled as graphs. In both cases, shortest path graph kernels are used to calculate similarities between proteins modeled as graphs.

[1] Authors also show that its runtime degenerates to 1 day for a dataset of 1178 examples with graphs having a maximum order of $|G| = 5748$

Chapter 4

Protein Function Prediction

This chapter deals with the application of graph kernels to the problem of protein function prediction. The methods and concepts presented in Chapters 2 and 3 are used to propose and evaluate a novel approach for exploiting protein structures in order to predict protein function. In particular, we present the results of experiments for two separate problems, namely, the prediction of proteins as enzymes and the recognition of DNA-binding proteins (DBPs). In both cases, the results achieved by our approach outperform other state-of-the-art methods.

Section 4.1 presents a brief introduction to basic molecular biology concepts that are relevant to the applications discussed in this chapter. For more technical elements, interested readers may refer to [53]. In Section 4.2, we outline the motivation behind our proposed novel methods for protein function prediction and an analytic summary of existing methods in the literature. Our approach is described in Section 4.3, wherein we introduce several graph models for protein structures. We explore three different strategies for modeling graphs. When protein graphs are created by clustering amino acid residues into vertices and labeling vertices with evolutionary profiles, the accuracy of protein function prediction improves by a large margin. With such a model fixed, we compare our approach against existing methods. Our results outperform others for the discrimination of proteins as enzymes (Section 4.4), and the recognition of DBPs (Section 4.5).

4.1 A Primer on Protein Structure

The most abundant macromolecules found within cells are *proteins*. They are composed of repeating structural units called *amino acids* connected by chemical bonds. Amino acids are molecules containing an amino group (NH_2) and a carboxyl group (COOH) bonded by

the α-*carbon* atom (also referred to as C-alpha atom). The α-carbon is also bonded to a hydrogen atom and to the R group. This group determines the identity of a particular amino acid.

Among all possible amino acids, only 20 are commonly found in proteins. They are known as the standard amino acids, and they are usually represented by three-letter abbreviations. A more compact one-letter code is also used to express sequences of amino acids. Both representations for the standard 20 amino acids are presented in Table 4.1.

Table 4.1: List of standard amino acids and their respective three-letter abbreviations and one-letter codes.

Alanine	Ala	A	Leucine	Leu	L
Arginine	Arg	R	Lysine	Lys	K
Asparagine	Asn	N	Methionine	Met	M
Aspartic acid	Asp	D	Phenylalanine	Phe	F
Cysteine	Cys	C	Proline	Pro	P
Glutamic acid	Glu	E	Serine	Ser	S
Glutamine	Gln	Q	Threonine	Thr	T
Glycine	Gly	G	Tryptophan	Trp	W
Histidine	His	H	Tyrosine	Tyr	Y
Isoleucine	Ile	I	Valine	Val	V

As the result of a chemical reaction between the carboxyl group of one molecule and the amino group of the other, by releasing a molecule of water, a pair of amino acids can be linked by a *peptide bond*. Polymers composed of many amino acids linked by such bonds are often referred to as *polypeptides*, which can be represented by ordered strings of characters from the alphabet of 20 one-letter codes. The portion of an amino acid that remains after a peptide bond formation is called *residue*. Figure 4.1[1] illustrates the reaction between two amino acids, forming a dipeptide.

The reactions described above are relevant because proteins are molecules that might contain one or more polypeptide chains. Due to their high degree of complexity, proteins are usually described in terms of four levels of organization: their primary, secondary, tertiary, and quaternary structure.

[1]Figure was released into the public domain at Wikimedia Commons

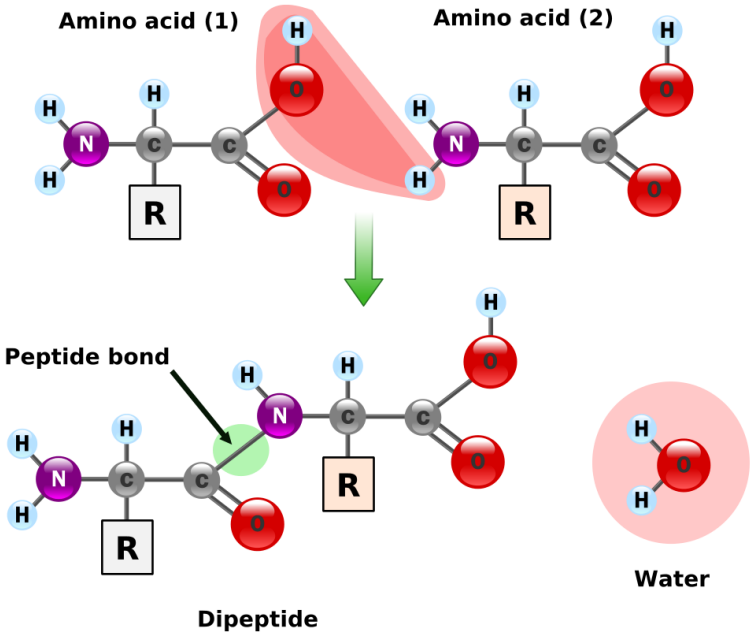


Fig. 4.1: Illustration of the formation of a peptide bond between two amino acids. The figure also shows that each amino acid has a C-alpha atom linking a hydrogen atom, and the amino, carboxyl and R groups.

Primary structure is a linear sequence of amino acids forming the polypeptide chain, or chains if the protein consists of more than one chain. The space of different possible sequences is astonishingly huge. For a protein of n residues, there are 20^n possible sequences.

Secondary structure is the arrangement of highly regular structures resulting from hydrogen bonding. Two types of secondary structure elements (SSEs) that frequently occur are the α-helix and the β-sheet.

Tertiary structure is the arrangement of all atoms in a 3-dimensional space. In general, this tertiary structure is determined by an experimental technique called *x-ray crystallography*.

Quaternary structure is a complex of several protein molecules, called subunits under this context.

The applications described in this chapter are based on modeling proteins as graphs, as described in Section 4.3. To this end, we use the tertiary structure of proteins, that is, the 3-dimensional structure of proteins. Indirectly, the primary structure is also considered for calculating evolutionary profiles, which are used as labels for vertices. On the other hand, we do not rely on secondary structure elements. They have been considered in [43]; however, experiments using our approach show better performance. Quaternary structure is not used because we focus on single proteins.

Information related to the different levels of organization of proteins can be freely downloaded from the protein data bank (PDB) [54]. The PDB archive is a centralized repository of protein structures from large biological molecules, including proteins and nucleic acids. Those molecules are found in all organisms including bacteria, yeast, plants, other animals, and humans.

4.2 Motivation

While genome sequencing projects have produced a large amount of protein sequences, several structural genomic projects have been targeting the large-scale determination of protein structures. Transforming such structural data into reliable insights about protein function and their underlying interaction patterns is still a major challenge. To enable researchers to keep pace with the growing number of structures, reliable and efficient computational methods that can reveal such patterns are critical.

The study of protein function, and particularly enzyme prediction, plays a central role in biology. For example, enzymes catalyze a series of chemical reactions occurring in cells, known as metabolic pathways. These are essential elements for understanding functional aspects of individual genes [55]. Therefore, accurate enzyme discrimination is an important step toward pathway prediction and functional annotation of genes. Moreover, the design of computational methods for enzyme discrimination can enable fast annotation of novel uncharacterized proteins, consequently, offering extensive advantages over experimental methods in terms of time and cost.

In general, earlier work on protein function prediction can be divided into three categories: (1) approaches based on sequence alignment, wherein new proteins are annotated with the same function associated with homologous proteins; (2) approaches based on structural alignment, wherein protein function is inferred based on structural similarity; and (3) approaches that compare proteins based on features extracted from sequence, structure, and other properties.

Function prediction using sequence comparison is based on the belief that similarity in sequence leads to similarity in function. Nevertheless, it is known that similar sequences can lead to different functions, and dissimilar sequences can also correspond to similar functions [56]. Despite the known disadvantages of this approach, according to Syed and Yona [55], sequence comparison is still the most commonly used method for function prediction. When sequence similarity is not sufficient for function prediction, researchers also rely on protein structure alignments. Structure alignments need to be considered carefully for function prediction, because a structural fold adopted by a given protein does not directly imply a function [57]. However, rather than performing sequence or structure alignments, some researchers represent proteins using local and global features extracted from proteins, and then apply machine learning methods to discover feature-function relationship for protein function prediction [57–60].

Particularly, earlier work on the discrimination of proteins as enzymes or not includes the method proposed by Dobson and Doig [58, 59]. They represented proteins as feature vectors computed from structures and used them for classification. Subsequently, Borgwardt et al. [43] modeled proteins as graphs using SSEs and introduced the use of graph kernels for classification. Likewise, we are motivated to model proteins as graphs because protein structures contain rich topological information that is important for protein function and should be preserved as much as possible.

Although graphs are a convenient way to represent structured objects, graphs cannot be easily used with common machine-learning methods. We explore different strategies of graph models for representing protein structures. We also integrate evolutionary profiles

into our graph models of proteins. Finally, we use shortest path graph kernels and a SVM classifier to discriminate proteins as enzymes.

4.3 Graph Models for Proteins

In our approach, every protein structure is modeled as a graph. To this end, we define the *protein graph*, as a labeled, undirected, and simple graph $G = \langle V, E \rangle$, where V is a set of vertices and E is a set of edges connecting pairs of vertices. Different from the methods described in [43,58], wherein SSEs are used as structural units to build the graphs, we model graphs by exclusively relying on the 3-dimensional coordinates of the protein C-alpha atoms.

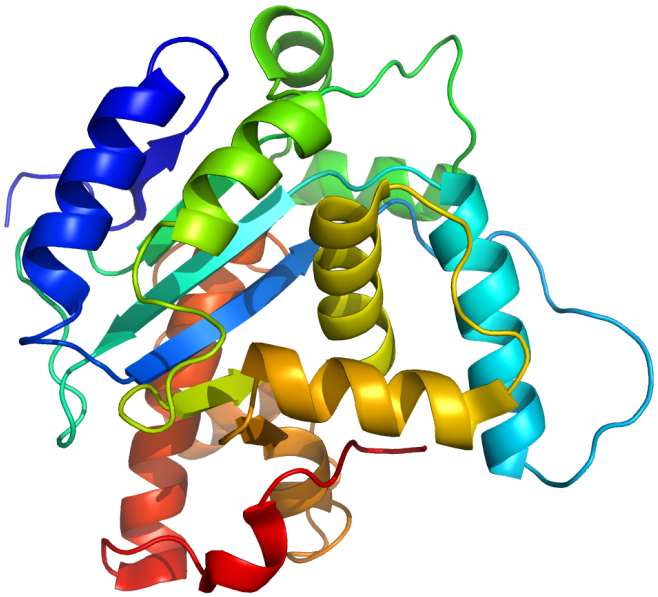
Given a protein structure, the idea is to partition their amino acid residues into different groups. Then, a vertex is created for each group of residues and is labeled accordingly. A pair of vertices $u, v \in V$ is connected by an edge $e \in E$ with weight $w = 1$ if the closest distance between atoms from the two vertices is less than $5\mathring{A}$. Residues are considered in contact when the minimum inter-residue distance of all pairs of atoms is less than $5\mathring{A}$. Figure 4.2[2] illustrates the reduction of a protein structure to an undirected graph with continuous labels at their nodes.

4.3.1 Partitioning Residues into Vertices

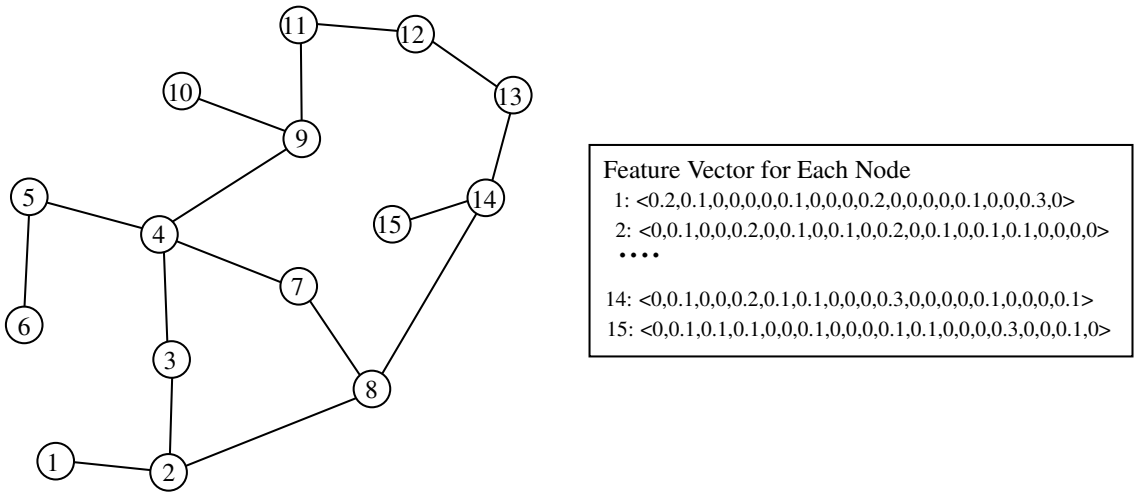
We explore three different strategies to partition amino acid residues into groups based on their C-alpha atoms: *binning*, *pca-binning*, and *clustering*. In binning, the smallest 3-dimensional enclosing box to include all C-alpha atoms of the protein is calculated. Then, according to an external parameter b, the box is partitioned into a 3-dimensional grid of b^3 equally sized bins. An amino acid is assigned to a bin if its C-alpha atom lies inside the bin. Finally, every non-empty bin is a vertex of the graph. Thus, the graph has a maximum of b^3 vertices. Figure 4.3 illustrates how binning can be applied to a set of points.

One advantage of binning is its simplicity. However, the resulting residue groups depend on the coordinate system being used. Rotating the protein structure leads to a different partition. In order to make binning more robust to transformations of protein structures,

[2]Figure on the left-hand side was released into the public domain at Wikimedia Commons



(a) Protein Structure



(b) Protein Graph

Fig. 4.2: Illustration of a protein graph. A protein structure (4.2(a)) is transformed into an undirected graph (4.2(b)) by grouping their atoms into vertices, and connecting pairs of contacting vertices. Every vertex in the graph is labeled with a feature vector.

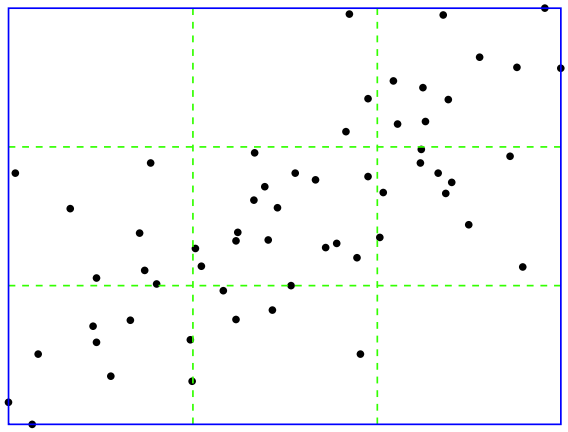
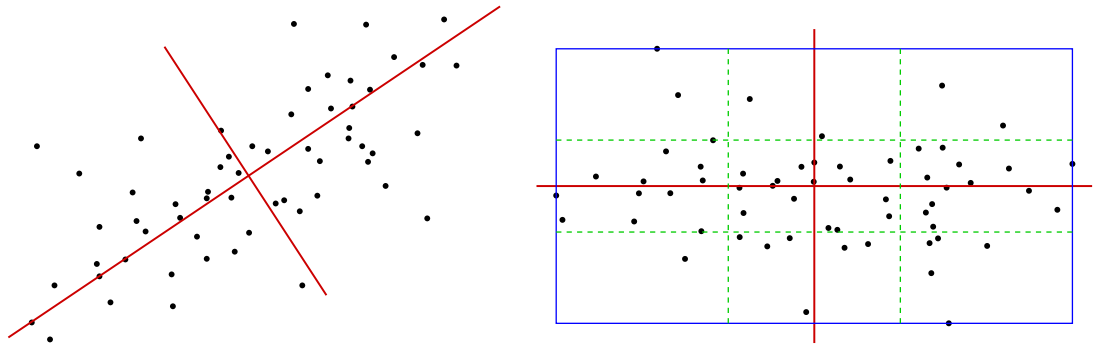


Fig. 4.3: Binning applied to a set of 2-dimensional points with parameter $b = 3$. In this example, the graph will contain only 7 vertices because two bins do not contain any points. Bear in mind that 2-dimensional points are used only for illustration purposes. In our work, the points lie on a 3-dimensional space.

we propose pca-binning. The set of C-alpha atoms is transformed into a new coordinate system by applying principal component analysis (PCA) and selecting all three principal components, which include all the information related to C-alpha atoms in the transformed domain. In this way, simple binning can be applied, and the partitioning is invariant to rotation. The application of pca-binning to the same set of points shown in Figure 4.3 is illustrated in Figure 4.4. Note how the resulting graph is completely different from merely applying binning on the original C-alpha atoms.

The third strategy explored to partition residues is *hierarchical agglomerative clustering* (HAC). We choose HAC because the resulting clusters after different runs remain the same, which is a desirable characteristic because there is a one-to-one relationship between a protein and its respective clustering partition. Some other traditional methods like k-means or a mixture of gaussians are not suitable because they yield different partitions after every run. Furthermore, by using HAC, the partition is invariant to protein rotation. Thus, it is not necessary to align all the protein 3-dimensional coordinates.

When using HAC, initially, each C-alpha constitutes an independent cluster. Then the number of clusters is reduced by hierarchically merging pairs of nearest clusters. We use the Ward's linkage method for merging clusters. This linkage method is known to be very



(a) (b)

Fig. 4.4: PCA-binning applied to the same set of points as in Figure 4.3. The calculation of the principal components is shown in Figure 4.4(a) and the application of simple binning to the transformed points is illustrated in Figure 4.4(b). Note how the points tend to be distributed in all the bins. A 2-dimensional space is used for illustration purposes. C-alpha atoms are in fact 3-dimensional.

robust when used with Euclidean distances [61], as in our case. The outcome of HAC is invariant to the rotation of protein structures. Figure 4.5 shows the clusters after applying HAC on a set of 12 amino acid residues.

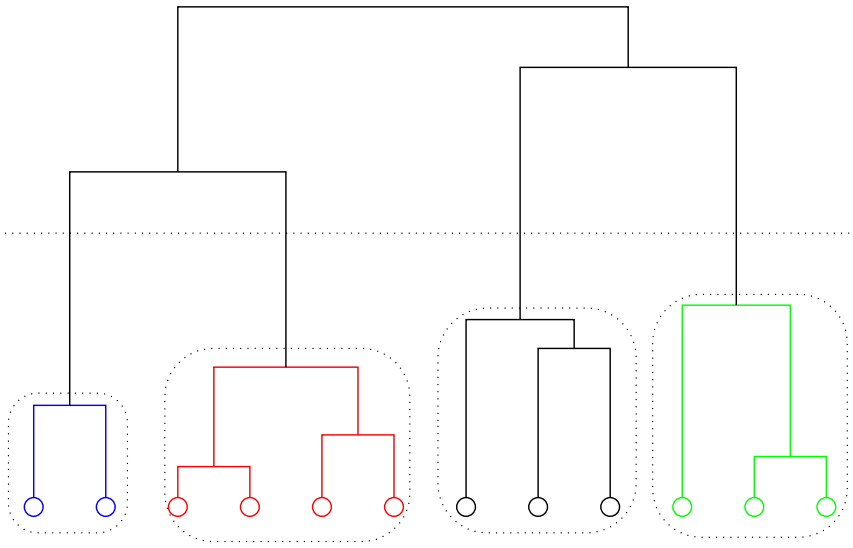


Fig. 4.5: Clustering applied to a set of 12 amino acid residues. For this illustration four clusters, shown in different colors within the four rounded rectangles, are found by the HAC algorithm. Note how HAC chooses neighboring points to form final clusters. Non-empty clusters become vertices in the graph.

4.3.2 Labeling Vertices

After the partitioning of amino acid residues has been generated, every non-empty group of residues becomes a vertex of the graph, and a feature vector is created at every vertex. To this end, we explored two types of labels for vertices, namely the *composition vector*, which is a common choice in the protein function prediction literature, and the *homologous vector*, which is one of our contributions. In this dissertation, the homologous vector is also referred to as the evolutionary profile.

Composition vector Each vertex is labeled with a normalized vector of 20 values that correspond to the frequency of the 20 types of amino acid residues in the vertex.

Homologous vector For each protein of interest, we performed 4 runs of PSI-BLAST [62] search to find homologous sequences from the NCBI non-redundant (nr) database, and built a multiple alignment. Then, each amino acid residue of the protein of interest was associated with a column of the multiple alignment. For each vertex, the residue composition was calculated by averaging all columns associated with the amino acids that belong to the vertex. The vertex was then labeled with the 20 values representing that residue composition. In this implementation, the vector not only takes into account the residue composition of the protein of interest, but also the residue composition of its homologous sequences.

4.3.3 Defining a Graph Kernel

Among all the graph kernels methods presented in Chapter 3, we use the shortest path class of graph kernels described in Section 3.3.4. This class allows the use of labeled graphs with continuous labels in polynomial time. We use the Floyd-Warshall method presented in Algorithm 3.1 for transforming graphs into shortest path graphs.

The kernel function presented in Eq. 3.10 requires the definition of kernels for vertices and edges in order to calculate the corresponding values of $k_{walk}(e, e')$. Following the notation of Eq. 3.11, we adapted the formulas presented in [43]. The kernel function for a

pair of vertices u and v is a Gaussian kernel over their respective feature vectors as shown in Eq. 4.1:

$$k_{node}(u,v) = \exp\left(-\frac{\|fv(u) - fv(v)\|^2}{2\delta^2}\right) \tag{4.1}$$

where δ is the kernel width, $fv(\cdot)$ denotes the feature vector (label) of a vertex, and $\|fv(u) - fv(v)\|$ represents the Euclidean distance between two feature vectors.

Similarly, we define the kernel function for a pair of edges e and f as a Brownian bridge kernel that assigns the highest value to edges with identical weights, and 0 to all edges that differ in weight more than a constant c, as shown in Eq. 4.2:

$$k_{edge}(e,f) = \max(0, c - |weight(e) - weight(f)|) \tag{4.2}$$

4.3.4 Embedding Graph Kernels into SVMs

SVMs are widely used in many computational biology problems due to their high accuracy, their ability to deal with large and high-dimensional datasets, and their flexibility in modeling diverse sources of data [7]. As demonstrated in Section 2.3.1, we can easily take advantage of the kernel trick by embedding the shortest path graph kernel in the dual formulation for SVMs as follows:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k_{sp}(x_i, x_j) \\ \text{subject to} \quad & 0 \le \alpha_i \le C, \qquad i = 1, \ldots, n \\ & \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned} \tag{4.3}$$

Likewise, the classification rule can be expressed in terms of a kernel between graphs as follows:

$$f(x) = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i^* y_i k_{sp}(x_i, x) + b^*\right). \tag{4.4}$$

In order to evaluate our proposed graph models and the use of shortest path graph kernels for protein function prediction, the following sections describe the experiments we

conducted with SVMs for two specific supervised learning problems: the discrimination of proteins as enzymes and the recognition of DBPs.

4.4 Enzyme Discrimination

This section describes the experiments conducted for enzyme discrimination. This is a binary classification problem, in which we evaluate the graph models described in Section 4.3 with the use of a shortest path kernel and the SVM classifier.

4.4.1 Dataset and General Settings

To evaluate our different strategies for representing proteins as graphs, we considered the same dataset used in previous work [43,58]. The original dataset consists of 1178 proteins with 691 (58.66%) enzymes and 487 (41.34%) non-enzymes. This dataset is challenging because no chain in any protein structurally aligns to any other chain with a Z-score of 3.5 or above.

Dobson and Doig [58] reported 76.86% accuracy in 10-fold cross-validation when using 52 features extracted from each protein, and 80.17% accuracy when using an optimized subset of features generated by adaptive search. Borgwardt et al. [43] considered a subset of the Dobson and Doig dataset. The subset consisted of 1128 proteins whose secondary structure is available at the PDB [54], with 59% of such proteins being enzymes. They reported 77.30% accuracy for their random walk graph kernel and 84.04% accuracy when global features were incorporated to vertices in the protein graph. We considered the $1,128$ proteins from Borgwardt's dataset. The structures were downloaded from PDB. Then, a total of 28 enzymes and 22 non-enzymes were discarded because their PDB files contain only C-alpha atoms. The fraction of enzymes remained at 58.77%, conserving the same proportion of positive examples.

In our experiments, we used 10-fold cross validation in order to find the best parameter settings for both graph kernel and SVM classifier. Specifically, for the edge kernel we used $c = 2$ while for the node kernel we select δ from the interval $[0.05, 0.25]$ with values increasing at the stepsize of 0.1. Finally, for the C parameter required by SVMs, we selected the best

value from the exponentially growing sequence of C values given by 2^x where $-1 \leq x \leq 9$ with an increasing stepsize of 0.1.

4.4.2 Evaluating the Use of Evolutionary Profiles

With the goal of evaluating the performance of graph kernels on graphs with evolutionary profiles at vertices, we computed accuracy, specificity, sensitivity and the Matthews correlation coefficient (MCC) after applying our approach to graphs generated using the binning strategy. Accuracy indicates the overall percentage of correct predictions, specificity measures the percentage of non-enzymes that have been correctly recognized, sensitivity is the percentage of enzymes that have been correctly predicted as enzymes, and MCC shows the overlapping between the predictions and the actual distribution. MCC is a value in the range of $[-1, +1]$, with $+1$ indicating perfect predictions, -1 indicating totally wrong predictions, and 0 represents random predictions. Table 4.2 shows all these indicators for evaluating the performance of classifiers trained with composition vectors and homologous vectors at vertices, where parameter b determines the number of bins in the partitioning of amino acid residues. In Table 4.2, we clearly confirm that the use of homologous vectors (evolutionary profiles) makes a significant positive impact on all indicators. For this reason, all the remaining experiments were carried out with graphs generated only with homologous vectors at their nodes.

Table 4.2: Performance for classifying proteins as enzymes versus non-enzymes using binning on graphs with composition vectors and homologous vectors. Parameter b determines the number of bins in the partitioning of amino acid residues (e.g., the number of bins equals to b^3).

	Composition		Homologous	
	$b = 2$	$b = 3$	$b = 2$	$b = 3$
Accuracy	0.7828	0.7944	**0.8431**	0.8413
Specificity	0.7011	0.7377	0.7849	**0.7977**
Sensitivity	0.8401	0.8341	**0.8839**	0.8718
MCC	0.5487	0.5761	**0.6773**	0.6720

4.4.3 Improving Performance with PCA

The binning strategy yields different partitioning if the same protein is rotated. This shortcoming is addressed by identifying the principal components of the C-alpha atoms prior to the partitioning of residues into vertices. We tested both approaches, binning and pca-binning on the same dataset, using evolutionary profiles at the nodes. Table 4.3 shows a clear advantage of pca-binning over binning. For pca-binning, the accuracy, specificity and MCC are higher when $b = 3$. Higher values of b have been explored as well, but the performance decreases when b becomes greater than 3. A possible explanation is that when b becomes too large, most vertices only contain a very small number of residues, which makes the resulting graph less useful in capturing the structural properties of the protein.

Table 4.3: Performance for classifying proteins as enzymes versus non-enzymes using *binning* and *pca-binning* strategies and evolutionary profiles. Parameter b determines the number of bins in the partitioning of amino acid residues.

	Binning		PCA-Binning	
	$b = 2$	$b = 3$	$b = 2$	$b = 3$
Accuracy	0.8431	0.8413	0.8580	**0.8617**
Specificity	0.7849	0.7977	0.8043	**0.8238**
Sensitivity	0.8839	0.8718	**0.8958**	0.8884
MCC	0.6773	0.6720	0.7070	**0.7146**

4.4.4 Introducing Clustering of Amino Acids

With the hope of finding expressive graphs of smaller order, we used the HAC method with different values for k, the number of clusters (or vertices in the graph), ranging from 8 to 32. The results are shown in Table 4.4. Note when $k = 24$, the classifier achieved best performance, reaching 87% accuracy.

4.4.5 Comparisons with Existing Methods

Finally, the different graph representation strategies proposed in this work are compared with previous work. Table 4.5 shows that the accuracy obtained by the proposed

Table 4.4: Performance statistics for classification of proteins as enzymes versus non-enzymes when hierarchical clustering is used to partition amino acid residues into groups. k indicates the number of clusters.

k	Accuracy	Specificity	Sensitivity	MCC
8	0.8502	**0.8172**	0.8734	0.6920
12	0.8484	0.7762	0.8990	0.6861
16	0.8492	0.7611	0.9110	0.6874
20	0.8581	0.8085	0.8929	0.7065
24	**0.8697**	0.7980	**0.9186**	**0.7307**
28	0.8555	0.7918	0.9003	0.7030
32	0.8556	0.8046	0.8913	0.7029

strategies outperforms that of previous methods, with hierarchical clustering achieving the best performance in terms of all measures.

Table 4.5: Accuracy for discriminating enzymes using ten-fold cross-validation. The strategies proposed in this project outperform those reported in previous work using the same dataset.

Method	Accuracy
Clustering (ours)	**86.97**
PCA Binning with 3 bins (ours)	86.17
Binning with 2 bins (ours)	84.31
Graph kernel [43] (with global info)	84.04
Optimized vector kernel [58]	80.17
Graph kernel [43]	77.30
DALI classifier [43]	75.07

4.5 Recognition of DNA Binding Proteins

DNA binding proteins are part of several crucial processes in cells, such as DNA transcription, replication and packing [63]. Their correct identification is of great importance. In this section, we describe our experiments for the recognition of DBPs. We create graphs from the 3-dimensional structure of proteins using clustering of amino acid residues. Then, evolutionary profiles are used to label vertices. The SVM classifier, in conjunction with a shortest path graph kernel, is used for evaluating our approach. When the results are

compared with other methods, our approach shows higher performance.

4.5.1 Datasets

We used three different datasets from previous studies. Two of them contain DNA-binding proteins (positive examples) and one contains non-DNA binding proteins (negative examples). The first dataset (P138) has 138 DNA-binding proteins and was created by Szilágyi and Skolnick [64]. The mutual sequence similarity between any two chains in this dataset is no more than 35%. The second dataset (P78) was created by Ahmad and Sarai [65]. P78 contains 78 DNA-binding proteins, with pairwise sequence identity less than 25%. For the negative dataset, we used 110 non-DNA-binding proteins (N110) created by Ahmad and Sarai [65].

4.5.2 Experiments

Two different experiments were performed. First, we trained and evaluated SVM classifiers using 10-fold cross validation with datasets P78 and N110. Then we repeated the same process using P138 and N110. We tried different values of k for clustering and explored both composition vectors and homologous vectors. As in the enzyme discrimination experiments, our method achieved better results when the homologous vector was used. However, different from enzyme identification, the method achieved best results with $k = 20$. When $k = 20$, the method (with homologous vector) achieved 95.73% accuracy in the experiment using datasets P78 and N110 and 92.33% accuracy in the experiment using P138 and N110. As a comparison, when $k = 24$, the method (with homologous vector) achieved 94.71% accuracy in the experiment using datasets P78 and N110 and 91.53% accuracy in the experiment using P138 and N110. The fact that different optimal values of k were found in enzyme discrimination and DNA-binding protein discrimination may suggest that different numbers of clusters should be used to capture the unique patterns of residue distribution in the space for different types of function. In other words, the amino acids' distribution on the structures may follow different patterns for different types of function.

In previous studies, Ahmad and Sarai [65] and Szilágyi and Skolnick [64] have used

datasets P78 and N110 to evaluate their methods. Szilgyi and Skolnick [64] only reported MCC for their study. Ahmad and Sarai did not report MCC in their original paper, Szilágyi and Skolnick recalculated the MCC for the Ahmad and Sarai method based on the published data when they compare the two methods [64]. In Table 4.6, we show performance statistics comparing our method with previous work.

Table 4.6: Comparison of our method with others using datasets P78 and N110.

Method	Accuracy (%)	Specificity (%)	Sensitivity (%)	MCC
Our method	**95.73**	**96.36**	**94.64**	**0.92**
Ahmad and Sarai [65]	83.90	87.00	80.80	0.68
Szilágyi and Skolnick [64]	–	–	–	0.79

Szilágyi and Skolnick [64], Nimrod et al. [63], and Langlois and Lu [66] have used datasets P138 and N110 to evaluate their methods. Table 4.7 shows the comparisons between our method and theirs using P138 and N110. Szilágyi and Skolnick [64] only reported MCC for their study and Nimrod et al. [63] did not report accuracy. Tables 4.6 and 4.7 show that our method outperforms the others. Specifically, when comparing to the best performance in prior studies on the same dataset, our method improves the MCC value from 0.79 to 0.92 on datasets P78 and N110 and from 0.80 to 0.85 on datasets P138 and N110, respectively.

Table 4.7: Comparison of our method with others using datasets P138 and N110.

Method	Accuracy (%)	Specificity (%)	Sensitivity (%)	MCC
Our method	**92.33**	**91.82**	**92.75**	**0.85**
Nimrod et al. [63]	–	90.00	90.00	0.80
Szilágyi and Skolnick [64]	–	–	–	0.74
Langlois and Lu [66]	85.90	80.90	89.90	0.75

4.6 Conclusions

In this chapter, three different strategies for representing protein structures were explored. The results obtained for protein function prediction in two specific applications

show that the proposed graph modeling and kernel design is very competitive for problems involving protein structures.

Using a simple binning strategy, our proposed approach achieves better performance than state-of-the-art methods in classifying enzymes versus non-enzymes. With pca-binning and clustering strategies, performance is not only better than using simple binning, but the methods are also invariant to rotation of protein structures.

We also explored two different types of vertex labels: a composition vector, that showed the frequencies of different types of amino acids in a vertex, and a homologous vector, that took into account the residue composition from homologous sequences. When the homologous vector was used to label vertices, the accuracy was significantly improved.

Chapter 5

Protein Semantic Similarity

As the gene ontology (GO) plays an increasingly important role in bioinformatics research, there has been great interest in developing objective and accurate methods for calculating semantic similarities between GO terms and, at a higher level of abstraction, semantic similarities between proteins. In this chapter, two semantic similarity methods for proteins are proposed and discussed. Such methods exploit the knowledge available at the GO, and they are based on graph models and graph kernels.

An introduction to the basic concepts related to the GO is given in Section 5.1. Then, semantic similarity methods for GO terms and proteins are presented progressively in Sections 5.2 and 5.3. All sections include relevant information about the current advances and challenges in the development of methods for calculating semantic similarity based on GO annotations. We conclude the chapter in Section 5.4, by presenting a summary and some thoughts on the directions for future research in this field.

5.1 Introduction

Researchers have made great effort to develop objective and accurate methods to calculate semantic similarities, which measure the degree of similarity between the meanings of pairs of objects. For example, in the natural language processing community, the problem of estimating the semantic similarity between concepts has been a central topic for several years. Numerous methods based on the WordNet ontology have been produced to estimate semantic similarities [67]. In recent years, ontologies have grown to be a popular topic in the biomedical research community, creating a demand for computational methods that can exploit their knowledge and hierarchical structure.

The semantic relationships represented in ontologies are the basic building blocks for

the calculation of semantic similarities between concepts. In the particular case of the GO, the underlying ontology for the methods presented in this chapter, semantic similarities between pairs of GO terms are expected to reflect their biological closeness. Moreover, semantic similarity methods can easily be extended to infer higher level semantic relationships. Considering the case for estimating protein semantic similarity. Similarity scores for a given protein pair at the protein level can be calculated by combining the pairwise semantic similarities for the GO terms associated with such proteins.

Methods for calculating similarities between GO terms, or between proteins, are a relevant topic in computational biology. They can be used in a broad range of applications, including: clustering of genes in pathways [68–71], prediction of protein-protein interactions [72], and the evaluation of similarity between gene products with respect to expression profiles [73], protein sequence [74–76], protein function [77], and protein family [78].

5.1.1 The Gene Ontology

The most successful effort for systematically describing current biological knowledge is the GO project [12], which maintains a dynamic, structured, precisely defined, and controlled vocabulary of terms for expressing the roles of gene products (biochemical material, either RNA or protein) across species. The GO is dynamic in the sense that its structure changes as more information is available. The GO consists of three different ontologies describing: 1) *biological processes* (BP), wherein a process often involves a chemical or physical transformation (e.g., cell growth); 2) *molecular functions* (MF), wherein functions are defined as the biochemical activity of gene products (e.g., enzymes); and 3) *cellular components* (CC), which refers to places in the cell where gene products are active (e.g., nuclear membranes). Each ontology is structured as a rooted directed simple graph, in which nodes (GO terms) are linked to each other through *"is-a," "part-of,"* or *"regulates"* relationships. Such organization enables the retrieval and visualization of biological knowledge at different levels of abstraction.

5.1.2 Gene Ontology Annotations

The annotation of gene products is the process of assigning ontology terms to gene products in order to describe their activities and localization. For example, the GOA project [13], at the European Bioinformatics Institute (EBI), aims to provide high-quality annotations to UniProt KnowledgeBase (UniProtKB) entries. GOA annotations are obtained from strictly controlled manual and electronic methods, wherein every association is supported by a distinct evidence source. A protein can be annotated with multiple GO terms from any of the three ontologies in the GO. Functional annotations of UniProtKB proteins currently consist of over 32 million annotations, which cover more than 4 million proteins [13].

5.1.3 Semantic Similarity based on the Gene Ontology

Database entries generated manually and electronically by the GO and GOA projects are available for a new generation of methods that aim to estimate relationships between proteins by exploiting their semantic closeness. A common procedure of these methods is to calculate semantic similarities between the annotating GO terms of any two given proteins and combine those pairwise semantic similarities to obtain an overall final semantic similarity score. Different methods have been used to combine pairwise GO term similarities in previous research [17,69,73,75,76,79]. A representative collection of methods for calculating the semantic similarity between GO terms has been reviewed in [80].

Most of those methods use the information content (IC) of the nearest common ancestor (NCA) or the most informative common ancestor (MICA) in order to quantify the amount of shared information between two GO terms. The NCA of two terms is the common ancestor of both terms that is located farthest from the root in the GO. The IC can be calculated from the frequency of annotation of GO terms in external resources, such as the GOA database. Note that external resources change as knowledge is updated (e.g., more annotations are included in GOA). Consequently, for the same pair of GO terms, their semantic similarity computed by these methods might change as the external resources evolve. Semantic similarities between GO terms should not be affected by such changes

in the external resources. Semantic similarities should not show the occurrence of biased results, which are caused by certain frequent annotations in popular research topics.

Some other methods infer GO term semantic similarities based on distance measures [81,82], e.g., counting the number of edges on the shortest path between the involved terms in the GO. One shortcoming of this approach is that the edges in the GO do not imply equal length in semantics. Although some methods tried to address this issue by assigning different weights to edges at different levels, they still suffer from the fact that GO terms at the same level do not necessarily have the same specificity.

Additional methods calculate the semantic similarity between gene products without considering the semantic similarity between GO terms. In such methods, a gene product is represented by a set or a vector of GO terms that annotate it. Then, the semantic similarity between gene products is calculated as the overlap between sets or the inner product of vectors [69,75]. However, those methods do not exploit the structure of the GO and thus ignore the relationship between GO terms.

5.2 A Semantic Similarity Algorithm for Proteins

In this section, we describe a semantic similarity algorithm (SSA) that depends exclusively on the GO. SSA was originally proposed in our earlier work with the intention of calculating semantic similarity between pairs of English words using the WordNet ontology [16]. Herein, we present a new version of SSA with an improved formulation adapted to the GO. One of the main advantages in our method is that it does not assume a consistent quantity of parent-child specialization across relations in the gene ontology, as most purely structural methods do [68,83,84]. Another advantage is that SSA can be easily extended to calculate semantic similarity between gene products. In this study, we use proteins as examples of gene products. But it is worth pointing out that the GO also contains functional annotations for other gene products like non-coding RNA.

Given a pair of input proteins, SSA creates a subgraph of the GO including all terms annotating both input proteins and their respective ancestors in the GO hierarchy. Then, SSA exploits the subgraph to calculate pairwise semantic similarities between the GO terms

annotating the input proteins. Finally, such pairwise semantic similarities are combined to obtain a semantic similarity score for the two proteins. The reliability of our approach is evaluated by comparing the resulting semantic similarities between proteins against functional similarities between proteins derived from expert annotations. In the following sections, we describe this approach step by step.

5.2.1 Building the Subgraph from the Gene Ontology

For a pair of input proteins, all their annotating GO terms are collected. Next, a subgraph (denoted as G_{sim}) of the GO is created. Each vertex of G_{sim} is a GO term that either annotates one or both of the input proteins or it is an ancestor of the annotating terms. Each edge corresponds to a relationship between the two adjacent terms. We only focus on the two most common types of relationships in the GO: "is-a" and "part-of." Note that, once the GO includes three different ontologies, the resulting subgraph is different depending on which ontology is being used.

5.2.2 Semantic Similarity Between Gene Ontology Terms

Given two annotating GO terms t_1 and t_2, the semantic similarity between them is calculated on their respective subgraph G_{sim} according to:

$$SSA(t_1, t_2) = \frac{sp_{sim}(t_1, t_2) + nca_{sim}(t_1, t_2) + ld_{sim}(t_1, t_2)}{3} \tag{5.1}$$

where sp_{sim} is the distance of the shortest path between t_1 and t_2 in G_{sim} converted into a similarity value, nca_{sim} is a score proportional to the depth of the nearest common ancestor (NCA) of t_1 and t_2 in G_{sim}, and ld_{sim} is a similarity score between the definitions of the two terms.

All terms in Eq. 5.1 return values in the range of 0 to 1; consequently, their average is also a value between 0 and 1. We have also explored different weighting strategies by analyzing the contributions and correlation of the three terms. No weighting strategy could achieve consistent improvement on all the datasets. That is, although some weighting

choices gave minor improvement on some datasets, they also achieved worst performance in other datasets. Additionally, we explored the feasibility of calculating $SSA(t_1, t_2)$ without considering the similarity score between the definitions of the two terms. In those experiments, $SSA(t_1, t_2)$ was just the average of sp_{sim} and nca_{sim}. The three terms in Eq. 5.1 are detailed in the following sections.

Shortest Distance

One challenge in calculating semantic similarities derived from ontologies is that ontology relationships do not reflect equal semantic distances between terms. In the case of the GO, edges closer to the root usually represent larger differences in function (or other properties) than edges farther away from the root. To address this challenge, we assign weights (in the range of $[0, 1]$) to edges using Eq. 5.2, so that edges closer to the root have higher weights.

$$weight(t_i, t_j) = 1 - \frac{depth(t_i) + depth(t_j)}{2 \cdot max} \tag{5.2}$$

where, t_i and t_j are the endpoint vertices (terms) of the edge, $depth(t_i)$ and $depth(t_j)$ are their corresponding depths in the graph, and max is the maximum depth in the respective ontology. We then define the length of a path as the sum over the weights of the edges on the path. In SSA, the length of the shortest path between t_1 and t_2 is first calculated and then converted into a similarity score using:

$$sp_{sim}(t_i, t_j) = \left(\frac{sp(t_i, t_j)}{max} - 1 \right)^2 \tag{5.3}$$

where $sp(t_i, t_j)$ is the length of the shortest path from node t_i to node t_j, and max is the maximum depth in the ontology. Note that in Eq. 5.3 we normalize the value of $sp(t_i, t_j)$ by max which is the maximum depth. It can be easily shown that after weighting edges according to Eq. 5.2, the (weighted) length of the longest path between two GO terms in the graph is less than or equal to the maximum $depth(max)$. Thus, the normalization provides values in the interval $[0, 1]$. Finally, to convert distances into similarity values, we

subtract the distance by 1 and apply a quadratic function. In this way, longer distances between two terms give lower scores of similarity. We have also tried a linear function (absolute function) instead of a quadratic function to convert distance into similarity. No improvement was observed.

Depth of NCA

The depth of the NCA is calculated using Eq. 5.4, which is the depth of the nearest term that is a common ancestor of t_i and t_j, normalized by the maximum depth of the corresponding ontology in GO; therefore, nca_{sim} is limited to the interval $[0, 1]$.

$$nca_{sim}(t_i, t_j) = \frac{depth_{nca}(t_i, t_j)}{max} \tag{5.4}$$

where $depth_{nca}(\cdot)$ simply returns the depth of the nearest common ancestor. Note that in this formula, the deeper the NCA is, the more similar the two terms are.

Similarity between the Definitions

In the GO, each term is associated with a definition, which is a text note that encloses a rich source of knowledge about the meaning of the term. For example, the definition of the term GO:0051210 (*isotropic cell growth*) is "The process by which a cell irreversibly increases in size uniformly in all directions. In general, a rounded cell morphology reflects isotropic cell growth." We define the long definition of a term as the union of the terms' name and its definition. In the case of GO:0051210, the corresponding long definition is: "isotropic cell growth. The process by which a cell irreversibly increases in size uniformly in all directions. In general, a rounded cell morphology reflects isotropic cell growth."

The similarity score between the long definitions of t_1 and t_2 is given by $ld_{sim}(t_1, t_2)$. First, we refine every term's long definition in the GO by removing common words (e.g., of, a, the, is, etc.) and applying the Porter algorithm [85] for stemming. Then, for each term, we create a long definition vector in an n-dimensional ontology space. The dimensionality of the ontology space is defined by n, the total number of unique stemmed words found in

the long definitions of all terms in the respective ontology. Each value in the long definition vector represents the tf-idf (term frequency-inverse document frequency) weight [86] for the corresponding word. This weight evaluates how important a word is to its long definition. A high tf-idf weight is reached by a high word frequency in the long definition and a low occurrence of the word in the collection of long definitions in the ontology. Finally, the similarity score between the long definition vectors of two terms is defined as their cosine similarity:

$$ld_{sim}(t_i, t_j) = \frac{l\vec{d}_i \cdot l\vec{d}_j}{\|l\vec{d}_i\| \cdot \|l\vec{d}_j\|} \tag{5.5}$$

where $l\vec{d}_i$ and $l\vec{d}_j$ are the long definition vectors for terms t_i and t_j, respectively.

5.2.3 Semantic Similarity Between Proteins

Although SSA had been designed for calculating semantic similarity between GO terms, we can combine term similarities to obtain the semantic similarity between proteins as follows. Let $A(P_k) = \{t_{k1}, t_{k2}, \dots\}$ be the set of non-redundant GO terms annotating protein P_k. Then, given two input proteins P_i and P_j with annotation sets $A(P_i) = \{t_{i1}, t_{i2}, \dots, t_{im}\}$ and $A(P_j) = \{t_{j1}, t_{j2}, \dots, t_{jn}\}$, we obtain the similarity matrix $M_{m \times n}$, where $M(a, b)$ is the semantic similarity between terms t_{ia} and t_{jb} given by $SSA(t_{ia}, t_{jb})$.

Based on this matrix of pairwise similarities, we can calculate the semantic similarity between P_i and P_j using three different methods described in previous research [68, 73, 74, 76, 78, 79], namely, the maximum, the average, and the best match average. The maximum method takes the maximum value from the similarity matrix M and considers it as the semantic similarity between the proteins. The average method calculates the average of all entries in M and assigns it to the semantic similarity between the proteins. The best match average method considers the maximum values in each row (row maxima) and each column (column maxima). The average of row maxima and the average of column maxima are first calculated. Then, the average of the two is the semantic similarity between the proteins. In other words, the semantic similarity computed by the best match average method is

denoted by sim_{bma} and calculated as follows:

$$sim_{bma}(P_i, P_j) = \frac{row_{max}(P_i, P_j) + col_{max}(P_i, P_j)}{2} \tag{5.6}$$

where

$$\begin{aligned} row_{max}(P_i, P_j) &= \frac{1}{m}\sum_{a=1}^{m} \max_{1\leq b\leq n} SSA(t_{ia}, t_{jb}) \\ col_{max}(P_i, P_j) &= \frac{1}{n}\sum_{b=1}^{n} \max_{1\leq a\leq m} SSA(t_{ia}, t_{jb}). \end{aligned} \tag{5.7}$$

Intuitively, the maximum value in a row (or column) corresponds to the best hit when a term annotating a protein is compared with all the terms annotating the other. The average over row maxima (row_{max}) reflects the best hits when comparing terms from the protein associated with the rows against the protein associated with the columns. Conversely, the average over column maxima (col_{max}) reflects the best hits when comparing terms from the protein associated with the columns against the protein associated with the rows. Thus, the score given by the best match average method (sim_{bma}) reflects the best scores in both directional comparisons.

5.2.4 Experiments

In this section, we describe the experiments conducted with the purpose of evaluating the performance of SSA. First, four different versions of SSA are defined and used to calculate semantic similarities for a dataset of proteins. Then, correlations are calculated between the resulting similarities and functional similarities derived from family annotations. Finally, based on the correlations, the SSA version with the higher performance is selected to be compared against existing methods.

Protein Functional Similarity Derived from Pfam Annotations

In our work, the reliability of SSA is evaluated by comparing the resulting semantic similarities against protein functional similarity derived from expert annotations. Functional similarities between proteins were derived from the Pfam database [87], following the

same approach described in [78]. Thus, let $F(P_i) = \{f_{i1}, f_{i2}, \ldots, f_{im}\}$ be the set of Pfam annotations for protein P_i and $F(P_j) = \{f_{j1}, f_{j2}, \ldots, f_{jn}\}$ be that of P_j. The functional similarity between the two proteins can then be estimated using the Jaccard coefficient between the two annotation sets as follows:

$$sim_{pfam}(P_i, P_j) = \frac{|F(P_i) \cap F(P_j)|}{|F(P_i) \cup F(P_j)|} \tag{5.8}$$

where the $\cap$ and $\cup$ operators denote, respectively, the resulting intersection and union sets between the Pfam annotation sets.

In our experiments, for a set of proteins, their pairwise semantic similarities were calculated using SSA. Separately, the pairwise functional similarities were calculated based on their Pfam annotation as mentioned above. As in [78], the Pearson's correlation coefficient (PCC) between the semantic similarities and functional similarities was used for performance comparisons. The values of PCC range from -1 to 1. In this context, higher values of PCC indicate better performance in the calculation of semantic similarity. The PCC is calculated as follows:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{5.9}$$

where X and Y are two variables with means $\bar{X}$ and $\bar{Y}$, respectively. In our case, such variables correspond to the semantic and the functional similarities.

Datasets

We downloaded the revision 1.723 of the GO, for which Table 5.1 shows a summary of statistics. For the set of proteins, we used the release 15.6 of UniprotKB/Swiss-Prot. On the other hand, we used the release 74.0 of GOA-Uniprot as the database for protein annotations. In our preliminary experiments, we explored a few variations of SSA. To compare the performance of different versions, we selected the top 500 proteins with the most total number of annotations in GOA-Uniprot and created subsets with the top 100,

200, 300, 400, and 500 most annotated proteins. These subsets are referred to as *T-100*, *T-200*, *T-300*, *T-400*, and *T-500* respectively. We ensured that all selected proteins existed in UniprotKB/Swiss-Prot and had at least one annotation from each of the three GO ontologies in GOA-Uniprot. On the other hand, we also made sure that the selected proteins had at least one Pfam-A annotation by checking the online service provided by the Pfam database.

Table 5.1: Statistics for the revision 1.723 of the gene ontology. For each of the ontologies we show respectively the number of GO terms, the number of "is-a" links, the number of "part-of" links, and the maximum depth.

Ontology	terms	"is-a" links	"part-of" links	max-depth
Biological Process	16,819	27,532	3,446	14
Molecular Function	8,628	10,079	3	14
Cellular Location	2,416	3,670	941	10

Combining Pairwise GO Term Similarities

We compared the three different methods for combining pairwise semantic similarities between GO terms to obtain a semantic similarity between a pair of proteins, namely the maximum (MAX), average (AVE), and best match average (BMA). We implemented a very simple version of SSA, which considered only "is-a" relationships when constructing the subgraph, and did not use the similarity score between the definitions of the two terms. Table 5.2 shows the results when the MF ontology was used to build the subgraph. The results show that BMA outperforms MAX and AVE in all datasets used. The same trend was observed when the BP and CC ontologies were respectively used to build the subgraph. We also repeated this experiment using different versions of SSA (see details in the next section), and the same trend was observed. Since BMA outperforms AVE and MAX, in all of the following experiments, BMA was chosen to combine semantic similarities between GO terms to obtain the semantic similarity between proteins.

Table 5.2: Correlations between semantic similarities and functional similarities for different combination methods. This table shows that BMA outperforms MAX and AVE for the MF ontology. The same trend was observed when the BP and the CC ontologies were used.

Dataset	MAX	AVE	BMA
T-100	0.41	0.45	**0.77**
T-200	0.24	0.33	**0.62**
T-300	0.25	0.25	**0.65**
T-400	0.23	0.23	**0.57**
T-500	0.21	0.21	**0.51**

Exploring Several Versions of SSA

In the GO, each term is associated with a text definition. To the best of our knowledge, no existing method, based on the GO, explores the use of term definitions in the calculation of semantic similarity. Thus, we are interested in validating whether the use of similarities between term definitions improves the performance or not. We are also interested in observing the contributions of the most common types of relationships, to be precise, "is-a" and "part-of." Here, if a type of relationship is taken into account, the edges corresponding to this type of relationship are included in the subgraph. In order to achieve our goals, we performed experiments using distinct versions of SSA, described in Table 5.3.

Table 5.3: Different versions of SSA used in our experiments.

Version	Relationships considered	Term definitions considered
SSAv1	"is-a"	No
SSAv2	"is-a" and "part-of"	No
SSAv3	"is-a"	Yes
SSAv4	"is-a" and "part-of"	Yes

We do not evaluate SSA versions for which only "part-of" relationships are considered, because they lead to poor performance and "is-a" relationships are vital to represent the semantic relations between GO terms. Moreover, in the MF ontology, as shown in Table 5.1, the distribution of "is-a" and "part-of" relationships differ drastically. Note also that in SSAv2 and SSAv4, both the edges corresponding to "is-a" relationships and the edges

corresponding to "part-of" relationships are included in the subgraph, while that in SSAv1 and SSAv3, only the edges corresponding to "is-a" relationships are included. For each version of SSA, we calculated pairwise semantic similarities between proteins. Then the correlation between their resulting semantic similarities and functional similarities derived from Pfam annotations was used to evaluate the performance of each different version. This step was repeated for each of the five datasets and for each of the three ontologies in the GO. Table 5.4 shows the results of experiments with all versions.

Table 5.4: Correlations between protein functional similarities derived from Pfam annotations and semantic similarities given by four different versions of SSA.

Version	Dataset	SSAv1	SSAv2	SSAv3	SSAv4
BP	T-100	0.79	0.79	**0.81**	0.80
	T-200	0.68	0.66	**0.70**	0.68
	T-300	0.73	0.70	**0.74**	0.73
	T-400	0.64	0.61	**0.66**	0.64
	T-500	0.59	0.56	**0.61**	0.59
MF	T-100	0.77	0.77	**0.78**	0.78
	T-200	0.62	0.62	**0.64**	0.64
	T-300	0.65	0.65	**0.66**	0.66
	T-400	0.57	0.57	**0.58**	0.58
	T-500	0.51	0.51	**0.53**	0.53
CC	T-100	0.53	0.51	**0.56**	0.54
	T-200	0.39	0.37	**0.41**	0.40
	T-300	0.43	0.41	**0.46**	0.44
	T-400	0.34	0.32	**0.36**	0.35
	T-500	0.30	0.29	**0.32**	0.31

SSAv3 yields higher correlations than SSAv1 across all datasets and all ontologies. The only difference between SSAv1 and SSAv3 is that SSAv3 takes into account the similarity between the definitions of GO terms and SSAv1 does not. Thus, we can conclude that taking into account the definitions of GO terms can improve the performance. This conclusion is also supported by comparing the results of SSAv2 with those of SSAv4. Table 5.4 also shows that the correlations achieved by SSAv2 are lower than or equal to those of SSAv1 while the correlations achieved by SSAv4 are lower than or equal to those of SSAv3. From this, we can conclude that the use of "part-of" relationships does not improve the performance.

When the MF ontology is used to build the graph, SSAv2 achieves the same results as SSAv1 while SSAv4 achieves the same results as SSAv3. This is because there are only three "part-of" relationships in the MF ontology, which is too few to affect the results. In comparison, there are 3446 "part-of" relationships in the BP ontology and 944 in the CC ontology.

When the dataset and the method are fixed, SSA achieves much lower correlations when CC ontology is used to build the subgraph than when BP or MF is used. One possible explanation is that the BP and MF ontologies describe gene products using terms associated with biological processes and molecular functions, both of which are directly related to protein function. Consequently, the semantic similarities calculated using these ontologies correlate better with functional similarities. In contrast, the CC ontology describes gene products using terms associated with cellular components, which are only indirectly related to protein function. Another possible cause of this is that proteins in the dataset have different numbers of annotations from the three ontologies. For example, in dataset *T-500*, the 500 proteins have a total of $16,248$ unique annotations from the BP ontology, $5,029$ unique annotations from the MF ontology and only $4,298$ unique annotations from the CC ontology. In Table 5.4, we can observe a general trend of decreasing performance from *T-100* to *T-500* (with the exception of *T-300*). Note that from *T-100* to *T-500*, the average number of annotations per protein decreases. These results suggest that having more annotations per protein in the dataset leads to more reliable functional similarity estimation. This claim is also supported by the results from [72].

Comparisons with Previous Methods

In this section, we compare SSA against previously published methods. We used the collaborative evaluation of GO-based semantic similarity measures (CESSM) online tool [88], developed by the XLDB research group at the University of Lisbon. For the purpose of comparison, CESSM provides a standard dataset consisting of $13,340$ pairs of proteins involving $1,039$ distinct proteins and implements 11 state-of-the-art semantic similarity methods, namely, simGIC [89], simUI [74], and the average, maximum and best-match

average combinations of three different term similarity methods proposed by Resnik [90], Lin [91], and Jiang and Conrath [92]. As a result, users can compare their methods with the 11 methods using the standard dataset.

In CESSM, sequence similarity is calculated using RRBS [74], which is a relative measure of sequence similarity based on BLAST bitscores. In the evaluation of a semantic similarity method, CESSM compares the pairwise semantic similarities between proteins given by the method with the pairwise sequence similarities between the proteins. CESSM uses resolution to evaluate how well semantic similarities match sequence similarities.

First, the semantic similarity is plotted against sequence similarity. Then, the data points are divided into sequence similarity intervals, with all intervals having an equal number of data points. Let y be the variable representing the averaged semantic similarity of the intervals and x be the variable representing the averaged sequence similarity of the intervals. The behavior of y versus x is modeled using two normal cumulative distribution functions (NCDF) as $y = a + b \cdot NCDF_1(x) + c \cdot NCDF_2(x)$. Finally, the resolution of the semantic similarity method is given by the sum of the scale factors, i.e., $b + c$. Based on the authors' definition, resolution is the relative intensity in which variations in the sequence similarity scale are translated into the semantic similarity scale. A higher resolution value means that the semantic similarity method has a higher capability to distinguish between different levels of protein function. Therefore, a method with higher resolution performs better than a method with lower resolution. A detailed description of resolution can be found in [74].

In addition to resolution, CESSM provides three different ways for evaluating a semantic similarity method. Specifically, it calculates correlations between the resulting semantic similarities and (1) functional similarities measured as sequence similarities, (2) functional similarities derived from enzyme commission (EC) classification, and (3) functional similarities derived from Pfam annotations.

Since the results presented in Table 5.4 confirmed that SSAv3 achieves better performance than other versions of SSA, we compare SSAv3 with the methods implemented in

CESSM. In this comparison, only the MF ontology is used, because, as stated in [76], the use of BP and CC ontologies yielded worse correlation between semantic similarity and sequence similarity. Table 5.5 presents the resolution scores achieved by applying different methods to the standard dataset provided by CESSM. It shows that SSA achieves the highest resolution, outperforming other previously published methods. In addition to the high resolution, SSA has the advantage that it does not rely on an external database of functional annotation observations in the calculation of semantic similarity. In comparison, all the other methods shown in Table 5.5 need such a database (in this case, the annotations in GOA). Note also that the MAX and AVE combination methods, in all cases, give worse scores than BMA. Table 5.5 also lists the correlation values computed by CESSM, when comparing semantic similarity with function similarity derived from EC classification and Pfam annotations, respectively. The correlation values between semantic and sequence similarities are not listed because their relationship is not linear [74]. Instead, resolution values discussed earlier are used to evaluate how well the semantic similarity matches sequence similarity.

Table 5.5: Resolution values obtained for sequence similarity and correlation values obtained for EC class and Pfam similarities for SSAv3 and other methods implemented in CESSM.

Algorithm	Resolution	EC Correlation	Pfam Correlation
SSA – BMA (ours)	**0.972**	0.631	0.590
simUI	0.967	0.637	0.618
Resnik – BMA	0.958	0.603	0.572
simGIC	0.956	0.622	**0.638**
Lin – BMA	0.571	**0.642**	0.564
Lin – AVE	0.451	0.423	0.447
Resnik – AVE	0.415	0.397	0.440
Resnik – MAX	0.381	0.454	0.182
Lin – MAX	0.248	0.453	0.180
Jiang & Conrath – BMA	0.241	0.561	0.491
Jiang & Conrath – AVE	0.175	0.341	0.332
Jiang & Conrath – MAX	0.084	0.362	0.128

5.3 Graph Kernels for Protein Semantic Similarity

In this section, we present a shortest path graph kernel for calculating the semantic similarity between gene products. In this approach, each gene product is represented as a graph that is an induced subgraph of the GO. Then, the shortest path graph kernel from Section 3.3.4 is used to calculate the semantic similarity between a pair of graphs. This method is intrinsic to the GO, i.e., it does not rely on external resources to calculate the semantic similarity. Thus, it does not have the same drawbacks as methods based on the IC of GO terms. At the same time, it uses a graph to explicitly explore the GO structure and exploit the relationship between GO terms. Graph matching is computationally expensive in general. However, we designed an instance of the shortest path graph kernel to efficiently calculate the similarity between graphs. Using a comprehensive evaluation benchmark provided by CESSM, we compare our approach with other state-of-the-art methods.

5.3.1 A Shortest Path Graph Kernel for Proteins

We model a protein using a subgraph of the ontology that consists of all the GO terms annotating the protein and their ancestors in the ontology. In this subgraph, each edge corresponds to a relationship between two terms in the ontology. Figure 5.1 shows the graph generated from the CC ontology for UniprotKB protein P17252.

We used a shortest-path graph kernel to compare two graphs as described in Section 3.3.4, as well as, using the Floyd-Warshall method from Algorithm 3.1 for transforming graphs into shortest path graphs. As the instantiation of shortest-path graph kernels involves the definition of kernels for vertices and edges, we define the vertex kernel as a Dirac kernel and the edge kernel as a Brownian bridge kernel. Both the Dirac and the Brownian bridge kernels are known to be positive definite [29].

From Section 3.3.4, given two shortest path graphs $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$, a shortest path graph kernel is defined as:

$$k_{sp}(G, G') = \sum_{e \in E} \sum_{e' \in E'} k_{walk}(e, e') \tag{5.10}$$

where $k_{walk}(e, e') = k_{node}(u, u') \cdot k_{edge}(e, e') \cdot k_{node}(v, v')$ for $e = (u, v)$ and $e' = (u', v')$. In

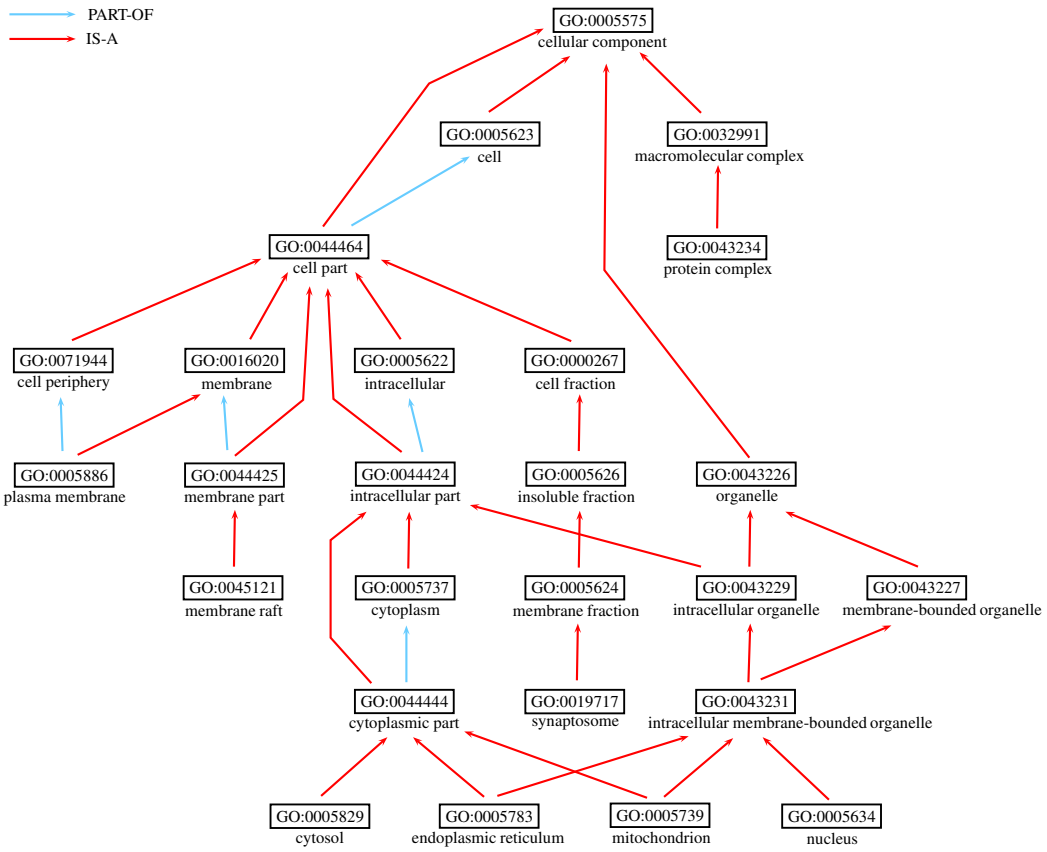


Fig. 5.1: Graph including all terms annotating protein P17252 (*protein kinase C alpha type*) from the cellular component ontology. All the ancestors of the annotating terms are also included in the graph.

our case, the kernel function for a pair of vertices is a Dirac kernel on their respective GO identifiers, formally defined as:

$$k_{node}(u, u') = \begin{cases} 1, & \text{if } id(u) == id(u') \\ 0, & \text{otherwise} \end{cases} \tag{5.11}$$

where the function $id(\cdot)$ simply returns the identifier for the GO term involved.

The kernel function for a pair of edges is a Brownian bridge kernel defined as follows:

$$k_{edge}(e, e') = \max(0, c - \left|weight(e) - weight(e')\right|) \tag{5.12}$$

where c is a constant. This kernel returns the largest value c, when two edges have identical weight, and 0 when the edges differ in weight more than c. In this study, we use $c = 2$.

5.3.2 Experiments

To compare our approach (referred to as sim_{spgk}) with other existing methods, we used the CESSM online tool [88], introduced in Section 5.2.4. As pointed out in [74], the maximum and average versions of term similarity methods have limitations from a biological point of view. Results presented in Table 5.5 also confirmed that the best-match average version has better performance than the maximum and average versions for Resnik [90], Lin [91], and Jiang and Conrath [92] methods. Thus, in this section, we compare sim_{spgk} with simGIC, simUI, and the best-match average versions of Resnik, Lin, and Jiang and Conrath methods. Bear in mind that the MF ontology is more closely related to function than the BP and CC ontologies; thus, we use the MF ontology to compare sim_{spgk} against different methods.

Table 5.6 shows the resolution values for different methods when sequence similarity is compared with semantic similarity. Tables 5.7 and 5.8 show the resulting Pearson's correlation coefficients for EC classification and Pfam annotations, respectively. To facilitate reading, we present the resolution and correlation values in separate tables, with algorithms

listed in descending order of performance.

Table 5.6: Resolution values for sim_{spgk} and other methods implemented in CESSM. Note also that sim_{spgk} yields higher resolution than SSA, described in Section 5.2.

Algorithm	Resolution
sim_{spgk} (ours)	**0.976**
SSA – BMA (ours)	0.972
simUI	0.967
Resnik – BMA	0.958
simGIC	0.956
Lin – BMA	0.571
Jiang & Conrath – BMA	0.241

The sim_{spgk} method achieves the best results in Tables 5.6 and 5.7, and it is the second best in Table 5.8. In addition to the better performance, the key advantage of sim_{spgk} is that it is intrinsic to the ontology, i.e., it does not rely on external resources in the calculation of the semantic similarity. In contrast, all the other methods, with the exception of simUI and SSA, rely on external resources, i.e., the annotations in GOA.

Table 5.7: Correlation values between semantic similarities and functional similarities derived from EC classification.

Algorithm	EC Correlation
sim_{spgk} (ours)	**0.646**
Lin – BMA	0.642
simUI	0.637
SSA – BMA (ours)	0.631
simGIC	0.622
Resnik – BMA	0.603
Jiang & Conrath – BMA	0.561

Among other semantic similarity methods listed in Table 5.6, simUI achieves the highest resolution. It computes the semantic similarity between two proteins as the ratio of the number of GO terms shared by the two proteins and the number of GO terms in their union. Thus, simUI requires only a linear time $O(n)$ and has the advantage that it is

simpler and faster for calculation. However, Tables 5.6, 5.7, and 5.8 show that sim_{spgk} outperformed simUI in all cases.

Table 5.8: Correlation values between semantic similarities and functional similarities derived from Pfam annotations.

Algorithm	Pfam correlation
simGIC	**0.638**
sim_{spgk} (ours)	0.622
simUI	0.618
SSA – BMA (ours)	0.590
Resnik – BMA	0.572
Lin – BMA	0.564
Jiang & Conrath – BMA	0.491

Despite the high computational cost involved in graph comparisons, sim_{spgk} does not suffer from this drawback. Using the shortest-path graph kernel, sim_{spgk} requires a polynomial time $O(n^4)$, where n is the number of vertices. In addition, each step of the graph kernel is simple to compute. For example, k_{node} only needs to compare whether two vertex IDs are identical since a Dirac kernel is used, and k_{edge} considers the length difference between two edges. Thus, the constant factors associated with the polynomial time complexity are very small, and sim_{spgk} can run fast in real applications.

5.4 Conclusion

This chapter presents two novel methods based on graph models and graph kernels, SSA and sim_{spgk} respectively, for the GO-based calculation of semantic similarity between proteins. In addition to their highly competitive performance, a major difference between them and other previous methods is that they rely exclusively on the GO. Furthermore, they do not assume a consistent quantity of parent-child specialization across relations in the gene ontology, as most purely structural methods do.

Particularly for SSA, other contributions include: (1) exploring the inclusion of definitions of GO terms in the semantic similarity calculation and confirmed that using such information can improve the performance; and (2) exploring the usefulness of "part-of"

and "is-a" relationships in the semantic similarity calculation and showing that taking into account "part-of" relationships does not help to improve the performance.

On the other hand, we show that our method based on graph kernels compares favorably with other state-of-the-art methods, including SSA. A big difference between sim_{spgk} and others is that sim_{spgk} takes into account the structure of the ontology. Since the structure of the ontology contains important information, it is beneficial to exploit it to capture semantic similarity. The results presented here show that sim_{spgk} provides an alternative to both methods that rely on external resources and intrinsic methods with comparable performance. Moreover, the use of graph kernels for comparing graphs has a very important advantage. It allows the direct embedding of semantic similarity measures into machine learning tasks, such as clustering and classification of gene products.

The evaluation of semantic similarity measures is especially difficult. In previous publications, the evaluation method was based on how well semantic similarity correlated with sequence similarity or functional similarity. Such an evaluation method assumes that there exist positive correlations between them, and the higher the correlation the better the measure. In this research, we follow the same assumptions made in previous studies. Although, it is probably true that there exists positive correlation between them, the claim that higher correlations mean better semantic similarity measures is still debatable. Further studies are still needed to test the validity of the assumptions.

Chapter 6

Conclusions

In this chapter, we present a summary of our work. In Section 6.1, we highlight our major contributions and present a list of publications associated with this dissertation. We show that our novel models for graphs in conjunction with shortest path graph kernels are suitable and effective for applications involving protein function prediction and protein semantic similarities. Section 6.2 gives an overview of some of the problems we want to address in our future work.

6.1 Summary

Recent technological advances have lead to an explosion of structured data, which are being produced at very high rates across a broad spectrum of disciplines and domains. In this context, graph kernels, which can be seen as similarity scores between graphs, appear as attractive and principled means to deal with discrete structures. They allow kernel based learning methods to be directly used on the data without resorting to vectorial representations. In this dissertation, we focus on the study of graph kernels and their applications to different computational biology problems.

The basic foundations of statistical learning theory, reviewed in Chapter 2, are the building blocks for understanding the kernel trick and support vector machines. Kernels are attractive because they allow the definition of similarity between pairs of objects, without imposing restrictions on the nature of the objects, as long as the kernels are valid with respect to symmetry and positive definiteness.

When the objects are graphs, the design of kernels faces two conflicting requirements. One is to find a representation that adequately captures the structural closeness of the input graphs. The other is to ensure the kernel is inexpensive to calculate. Existing graph

kernels, surveyed in Chapter 3, have been proposed in the literature to address this conflict. However, as a rule of thumb, increasing the expressivity of kernels demands higher, possibly exponential, computational time. Interestingly, the class of shortest path graph kernels provides a representation for graphs that is able to retain expressivity in polynomial time. Furthermore, instances of shortest path graph kernels are widely applicable because they do not impose limitations on the graph structure and they allow for labeling of vertices and edges.

Based on statistical learning theory and the recent advances in graph kernels, this dissertation presents contributions to the solution of two specific problems in bioinformatics. In Chapter 4, we propose novel models for protein structures, where compact graphs are created after applying hierarchical agglomerative clustering to the C-alpha atoms of the amino acid residues in the protein backbone. We also propose the use of evolutionary profiles at the vertices of each graph. We evaluate our approach under the context of protein function prediction. In particular, we use an instance of the class of shortest path graph kernels in conjunction with the SVM classifier, for solving the discrimination of proteins as enzymes and the recognition of DNA binding proteins. In both cases, the results were favorable to our approach. Preliminary results were presented in a conference [14] and extensions to this work were accepted for publication in a journal [15].

- Alvarez, M. A. and Yan, C. Exploring structural modeling of proteins for kernel-based enzyme discrimination. In *Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pp. 1–5, Montreal, Canada, 2010

- Alvarez, M. A. and Yan, C. A new protein graph model for function prediction. *Computational Biology and Chemistry*, 2012, to appear

In Chapter 5, we presented our findings for the problem of estimating semantic similarity between proteins. For a given pair of proteins, we explored the semantic relationships between their respective GO annotations. In previous work, we presented a novel algorithm

for semantic similarities between pairs of English words [16, 93]. This algorithm was extended for inferring the similarity of GO terms from the Gene Ontology, and subsequently refined to calculate the semantic similarity between proteins [17, 18].

Additionally, with the purpose of exploiting the rich structural information in the semantic space of the gene ontology, we proposed an instance of a shortest path graph kernel for calculating the semantic similarity of proteins [19]. To the best of our knowledge, this is the first attempt to use graph kernels for estimating semantic similarity. The results presented in Chapter 5 confirm that our proposed measures are highly competitive with standard techniques. Moreover, the use of graph kernels has an important impact in the estimation of semantic similarity based on ontologies, because, it turns the similarity measure into a positive definite function, allowing its use in powerful machine learning methods. The following is a list of publications derived from this work.

- Alvarez, M. A. and Lim, S. A graph modeling of semantic similarity between words. In *International Conference on Semantic Computing (ICSC)*, pp. 355–362, Irvine, CA, USA, 2007

- Alvarez, M. A., Qi, X., and Yan, C. Go-based term semantic similarity. In *Ontology Learning and Knowledge Discovery Using the Web: Challenges and Recent Advances*, ch. IX. IGI Publishing, 2011

- Alvarez, M. A. and Yan, C. A graph-based semantic similarity measure for the gene ontology. *Journal of Bioinformatics and Computational Biology*, 2011

- Alvarez, M. A., Qi, X., and Yan, C. A shortest-path graph kernel for estimating gene product semantic similarity. *Journal of Biomedical Semantics*, vol. 2, no. 1, p. 3, 2011

6.2 Extensions and Future Work

In general, most of the kernel functions described in Chapter 3 are derived from convolution kernels. That is, they are based on matching simple sub-structures like walks, trees, or cyclic patterns. Even though attractive kernels have been proposed under this

framework, challenges still remain for the development of more expressive kernels that can capture the semantics involved in graph similarities and are reasonably efficient to compute.

In such a context, possible avenues for further work are: 1) the exploration of new graph kernels for the efficient evaluation of graph similarities; 2) the investigation of graph signatures, which can provide more efficient algorithms not only for calculating graph similarities, but also for generating new graphs; 3) the innovative application of graph similarities and kernels in complex domains. For example, systems researchers and developers face challenges to keep pace with microprocessor evolution. Meaningful representation of programs structured as graphs can leverage novel machine learning applications such as automatic tuning of optimizations to be applied to a program [94], or the identification of vulnerabilities in both binary and source code.

References

[1] Snijders, T. Statistical models for social networks. *Annual Review of Sociology*, vol. 37, no. 1, pp. 131–153, 2011.

[2] Mihalcea, R. and Radev, D. *Graph-Based Natural Language Processing and Information Retrieval*. Cambridge University Press, 2011.

[3] Kashima, H., Saigo, H., Hattori, M., and Tsuda, K. Graph kernels for chemoinformatics. In *Chemoinformatics and Advanced Machine Learning Perspectives*, ch. I. IGI Publishing, 2011.

[4] Larrañaga, P., Calvo, B., Santana, R., Bielza, C., Galdiano, J., Inza, I., Lozano, J., Armañanzas, R., Santafé, G., Pérez, A., and Robles, V. Machine learning in bioinformatics. *Briefings in Bioinformatics*, vol. 7, no. 1, pp. 86–112, 2006.

[5] Vishwanathan, S. V. N., Kaski, S., Neville, J., and Wrobel, S. Introduction to the special issue on mining and learning with graphs. *Machine Learning*, vol. 82, pp. 91–93, 2011.

[6] Watson, J. and Thornton, J. Protein function prediction from structure in structural genomics and its contribution to the study of health and disease. In *From Molecules to Medicines*, pp. 201–215. Springer, 2009.

[7] Ben-Hur, A., Ong, C., Sonnenburg, S., Schölkopf, B., and Rätsch, G. Support vector machines and kernels for computational biology. *PLoS Computational Biology*, vol. 4, no. 10, pp. e1 000 173+, 2008.

[8] Müller, K., Mika, S., Ratsch, G., Tsuda, K., and Schölkopf, B. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181–201, 2001.

[9] Schölkopf, B., Tsuda, K., and Vert, J. P. *Kernel Methods in Computational Biology*. MIT Press, 2004.

[10] Borgwardt, K. M. and Kriegel, H. P. Shortest-path kernels on graphs. In *IEEE International Conference on Data Mining (ICDM)*, pp. 74–81, 2005.

[11] Staab, S. and Studer, R. *Handbook on Ontologies*. Springer, 2009.

[12] Ashburner, M., Ball, C., Blake, J., Botstein, D., Butler, H., Cherry, J., Davis, A., Dolinski, K., Dwight, S., Eppig, J., Harris, M., Hill, D., Tarver, I., Kasarskis, A., Lewis, S., Matese, J., Richardson, J., Ringwald, M., Rubin, G., and Sherlock, G. Gene ontology: Tool for the unification of biology. the gene ontology consortium. *Nature Genetics*, vol. 25, no. 1, pp. 25–29, 2000.

[13] Barrell, D., Dimmer, E., Huntley, R., Binns, D., O'Donovan, C., and Apweiler, R. The goa database in 2009 – an integrated gene ontology annotation resource. *Nucleic Acids Research*, vol. 37, no. suppl 1, pp. D396–D403, 2009.

[14] Alvarez, M. A. and Yan, C. Exploring structural modeling of proteins for kernel-based enzyme discrimination. In *Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pp. 1–5, Montreal, Canada, 2010.

[15] Alvarez, M. A. and Yan, C. A new protein graph model for function prediction. *Computational Biology and Chemistry*, 2012, to appear.

[16] Alvarez, M. A. and Lim, S. A graph modeling of semantic similarity between words. In *International Conference on Semantic Computing (ICSC)*, pp. 355–362, Irvine, CA, USA, 2007.

[17] Alvarez, M. A., Qi, X., and Yan, C. Go-based term semantic similarity. In *Ontology Learning and Knowledge Discovery Using the Web: Challenges and Recent Advances*, ch. IX. IGI Publishing, 2011.

[18] Alvarez, M. A. and Yan, C. A graph-based semantic similarity measure for the gene ontology. *Journal of Bioinformatics and Computational Biology*, 2011.

[19] Alvarez, M. A., Qi, X., and Yan, C. A shortest-path graph kernel for estimating gene product semantic similarity. *Journal of Biomedical Semantics*, vol. 2, no. 1, p. 3, 2011.

[20] Wang, J., Zhou, X., Zhu, J., Zhou, C., and Guo, Z. Revealing and avoiding bias in semantic similarity scores for protein pairs. *BMC Bioinformatics*, vol. 11, no. 1, p. 290, 2010.

[21] Vapnik, V. *The Nature of Statistical Learning Theory.* Springer–Verlag, 1995.

[22] Von Luxburg, U. and Schölkopf, B. Statistical learning theory: Models, concepts, and results. In *Inductive Logic*, ser. Handbook of the History of Logic, vol. 10, pp. 651–706, 2011.

[23] Cortes, C. and Vapnik, V. Support vector networks. *Machine Learning*, vol. 20, pp. 273–297, 1995.

[24] Haykin, S. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1998.

[25] Evgeniou, T., Pontil, M., and Poggio, T. Statistical learning theory: A primer. *International Journal of Computer Vision*, vol. 38, pp. 9–13, 2000.

[26] Schölkopf, B. Statistical learning theory, capacity, and complexity. *Complexity*, vol. 8, no. 4, pp. 87–94, 2003.

[27] Schölkopf, B. and Smola, A. A short introduction to learning with kernels. In *Advanced Lectures on Machine Learning*, pp. 41–64, 2003.

[28] Burges, C. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, 1998.

[29] Schölkopf, B. and Smola, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2001.

[30] Vert, J. P., Tsuda, K., and Schölkopf, B. A primer on kernel methods. In *Kernel Methods in Computational Biology*, pp. 35–70, 2004.

[31] Diestel, R. *Graph Theory.* Springer–Verlag, 2010.

[32] Conte, D., Foggia, P., Sansone, C., and Vento, M. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 18, pp. 265–298, 2004.

[33] Neuhaus, M. and Bunke, H. *Bridging the Gap Between Graph Edit Distance and Kernel Machines.* World Scientific Publishing Company, 2007.

[34] Gao, X., Xiao, B., Tao, D., and Li, X. A survey of graph edit distance. *Pattern Analysis & Applications*, vol. 13, pp. 113–129, 2010.

[35] Luo, B., Wilson, R., and Hancock, E. Spectral embedding of graphs. *Pattern Recognition*, vol. 36, no. 10, pp. 2213–2230, 2003.

[36] Wilson, R., Hancock, E., and Luo, B. Pattern vectors from algebraic graph theory. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 1112–1124, 2005.

[37] Bunke, H. and Riesen, K. Towards the unification of structural and statistical pattern recognition. *Pattern Recognition Letters*, 2011, to appear.

[38] Riesen, K. and Bunke, H. *Graph Classification and Clustering Based on Vector Space Embedding.* World Scientific Publishing Company, 2010.

[39] Haussler, D. Convolution kernels on discrete structures. UC Santa Cruz, Tech. Rep. UCSC-CRL-99-10, 1999.

[40] Vishwanathan, S. V. N., Schraudolph, N., Kondor, R., and Borgwardt, K. Graph kernels. *Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.

[41] Gärtner, T., Flach, P., and Wrobel, S. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, ser. Lecture Notes in Computer Science, vol. 2777, pp. 129–143, 2003.

[42] Kashima, H., Tsuda, K., and Inokuchi, A. Marginalized kernels between labeled graphs. In *International Conference in Machine Learning (ICML)*, pp. 321–328, 2003.

[43] Borgwardt, K., Ong, C., Schönauer, S., Vishwanathan, S., Smola, A., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, vol. 21, no. suppl 1, pp. i47–i56, 2005.

[44] Mahé, P., Ueda, N., Akutsu, T., Perret, J., and Vert, J. Extensions of marginalized graph kernels. In *International Conference on Machine Learning (ICML)*, pp. 552–559, 2004.

[45] Horváth, T., Gärtner, T., and Wrobel, S. Cyclic pattern kernels for predictive graph mining. In *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 158–167, 2004.

[46] Ramon, J. and Gärtner, T. Expressivity versus efficiency of graph kernels. In *International Workshop on Mining Graphs, Trees and Sequences*, pp. 65–74, 2003.

[47] Shervashidze, N. and Borgwardt, K. Fast subtree kernels on graphs. In *Neural Information Processing Systems Conference (NIPS)*, pp. 1660–1668, 2009.

[48] Mahé, P. and Vert, J.-P. Graph kernels based on tree patterns for molecules. *Machine Learning*, vol. 75, pp. 3–35, 2009.

[49] Floyd, R. Algorithm 97: Shortest path. *Communications of the ACM*, vol. 5, pp. 345+, 1962.

[50] Fröhlich, H., Wegner, J., Sieker, F., and Zell, A. Optimal assignment kernels for attributed molecular graphs. In *International Conference on Machine Learning (ICML)*, pp. 225–232, 2005.

[51] Gaüzère, B., Brun, L., and Villemin, D. Two new graph kernels and applications to chemoinformatics. In *Workshop on Graph-Based Representations in Pattern Recognition (GbR)*, pp. 112–121, 2011.

[52] Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. *Journal of Machine Learning Research (JMLR) – Proceedings Track*, vol. 5, pp. 488–495, 2009.

[53] Voet, D., Voet, J., and Pratt, C. *Fundamentals of Biochemistry: Life at the Molecular Level*. Wiley, 2008.

[54] Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P. The protein data bank. *Nucleic Acids Research*, vol. 28, no. 1, pp. 235–242, 2000.

[55] Syed, U. and Yona, G. Enzyme function prediction with interpretable models. In *Computational Systems Biology*, vol. 541, pp. 373–420, 2009.

[56] Hegyi, H. and Gerstein, M. The relationship between protein structure and function: A comprehensive survey with application to the yeast genome. *Journal of Molecular Biology*, vol. 288, no. 1, pp. 147–164, 1999.

[57] Lee, D., Redfern, O., and Orengo, C. Predicting protein function from sequence and structure. *Nature Reviews Molecular Cell Biology*, vol. 8, no. 12, pp. 995–1005, 2007.

[58] Dobson, P. and Doig, A. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, vol. 330, no. 4, pp. 771–783, 2003.

[59] Dobson, P. and Doig, A. Predicting enzyme class from protein structure without alignments. *Journal of Molecular Biology*, vol. 345, no. 1, pp. 187–199, 2005.

[60] Pandey, G., V., K., and Steinbach, M. Computational approaches for protein function prediction: A survey. Department of Computer Science and Engineering, University of Minnesota, Twin Cities, Tech. Rep. 06-028, 2006.

[61] Scheibler, D. and Schneider, W. Monte-carlo tests of the accuracy of cluster analysis algorithms – a comparison of hierarchical and non-hierarchical methods. *Multivariate Behavorial Research*, vol. 20, no. 3, pp. 283–304, 1985.

[62] Altschul, S., Madden, T., Schäffer, A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. Gapped blast and psi-blast: A new generation of protein database search programs. *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[63] Nimrod, G., Szilágyi, A., Leslie, C., and Ben-Tal, N. Identification of dna-binding proteins using structural, electrostatic and evolutionary features. *Journal of Molecular Biology*, vol. 387, no. 4, pp. 1040–1053, 2009.

[64] Szilágyi, A. and Skolnick, J. Efficient prediction of nucleic acid binding function from low-resolution protein structures. *Journal of Molecular Biology*, vol. 358, no. 3, pp. 922–933, 2006.

[65] Ahmad, S. and Sarai, A. Moment-based prediction of dna-binding proteins. *Journal of Molecular Biology*, vol. 341, no. 1, pp. 65–71, 2004.

[66] Langlois, R. and Lu, H. Boosting the prediction and understanding of DNA-binding domains from sequence. *Nucleic Acids Research*, vol. 38, no. 10, pp. 3149–3158, 2010.

[67] Budanitsky, A. and Hirst, G. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, vol. 32, no. 1, pp. 13–47, 2006.

[68] Wang, J., Du, Z., Payattakool, R., Yu, P., and Chen, C.-F. A new method to measure the semantic similarity of go terms. *Bioinformatics*, vol. 23, no. 10, pp. 1274–1281, 2007.

[69] Sheehan, B., Quigley, A., Gaudin, B., and Dobson, S. A relation based measure of semantic similarity for gene ontology annotations. *BMC Bioinformatics*, vol. 9, no. 1, pp. 468+, 2008.

[70] Nagar, A. and Al-Mubaid, H. A new path length measure based on go for gene similarity with evaluation using sgd pathways. In *IEEE International Symposium on Computer-Based Medical Systems*, pp. 590–595, 2008.

[71] Du, Z., Li, L., Chen, C.-F., Yu, P., and Wang, J. G-sesame: Web tools for go-term-based gene similarity analysis and knowledge discovery. *Nucleic Acids Research*, vol. 37, pp. W345–W349, 2009.

[72] Xu, T., Du, L., and Zhou, Y. Evaluation of go-based functional similarity measures using s. cerevisiae protein interaction and expression profile data. *BMC Bioinformatics*, vol. 9, no. 1, pp. 472+, 2008.

[73] Sevilla, J., Segura, V., Podhorski, A., Guruceaga, E., Mato, J., Martínez-Cruz, L., Corrales, F., and Rubio, A. Correlation between gene expression and go semantic similarity. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 2, no. 4, pp. 330–338, 2005.

[74] Pesquita, C., Faria, D., Bastos, H., Ferreira, A., Falcão, A., and Couto, F. Metrics for go based protein semantic similarity: A systematic evaluation. *BMC Bioinformatics*, vol. 9, no. Suppl 5, p. S4, 2008.

[75] Mistry, M. and Pavlidis, P. Gene ontology term overlap as a measure of gene functional similarity. *BMC Bioinformatics*, vol. 9, no. 1, pp. 327+, 2008.

[76] Lord, P., Stevens, R., Brass, A., and Goble, C. Investigating semantic similarity measures across the gene ontology: The relationship between sequence and annotation. *Bioinformatics*, vol. 19, no. 10, pp. 1275–1283, 2003.

[77] Fontana, P., Cestaro, A., Velasco, R., Formentin, E., and Toppo, S. Rapid annotation of anonymous sequences from genome projects using semantic similarities and a weighting scheme in gene ontology. *PLoS ONE*, vol. 4, no. 2, pp. e4619+, 2009.

[78] Couto, F., Silva, M., and Coutinho, P. Measuring semantic similarity between gene ontology terms. *Data & Knowledge Engineering*, vol. 61, no. 1, pp. 137–152, 2007.

[79] Schlicker, A., Domingues, F., Rahnenfuhrer, J., and Lengauer, T. A new measure for functional similarity of gene products based on gene ontology. *BMC Bioinformatics*, vol. 7, no. 1, pp. 302+, 2006.

[80] Pesquita, C., Faria, D., Falcão, A., Lord, P., and Couto, F. Semantic similarity in biomedical ontologies. *PLoS Computational Biology*, vol. 5, no. 7, p. 12, 2009.

[81] Cheng, J., Cline, M., Martin, J., Finkelstein, D., Awad, T., Kulp, D., and Siani-Rose, M. A knowledge-based clustering algorithm driven by gene ontology. *Journal of Biopharmaceutical Statistics*, vol. 14, no. 3, pp. 687–700, 2004.

[82] Wu, X., Zhu, L., Guo, J., Zhang, D.-Y., and Lin, K. Prediction of yeast proteinprotein interaction network: Insights from the gene ontology and annotations. *Nucleic Acids Research*, vol. 34, no. 7, pp. 2137–2150, 2006.

[83] Ruths, T., Ruths, D., and Nakhleh, L. Gs2: An efficiently computable measure of go-based similarity of gene sets. *Bioinformatics*, vol. 25, no. 9, pp. 1178–1184, 2009.

[84] Chabalier, J., Mosser, J., and Burgun, A. A transversal approach to predict gene product networks from ontology-based similarity. *BMC Bioinformatics*, vol. 8, no. 1, pp. 235+, 2007.

[85] Porter, M. An algorithm for suffix stripping. *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[86] Robertson, S. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, pp. 503–520, 2004.

[87] Finn, R., Tate, J., Mistry, J., Coggill, P., Sammut, S., Hotz, H.-R., Ceric, G., Forslund, K., Eddy, S., Sonnhammer, E., and Bateman, A. The pfam protein families database. *Nucleic Acids Research*, vol. 36, no. suppl 1, pp. D281–D288, 2008.

[88] Pesquita, C., Pessoa, D., Faria, D., and Couto, F. Cessm: Collaborative evaluation of semantic similarity measures. In *JB2009: Challenges in Bioinformatics*, 2009.

[89] Pesquita, C., Faria, D., Bastos, H., Falcão, A., and Couto, F. Evaluating go-based semantic similarity measures. In *Annual Bio-Ontologies Meeting*, pp. 37–40, 2007.

[90] Resnik, P. Using information content to evaluate semantic similarity in a taxonomy. In *International Joint Conference on Artificial intelligence (IJCAI)*, pp. 448–453, 1995.

[91] Lin, D. An information-theoretic definition of similarity. In *International Conference on Machine Learning (ICML)*, pp. 296–304, 1998.

[92] Jiang, J. and Conrath, D. Semantic similarity based on corpus statistics and lexical taxonomy. In *International Conference on Research in Computational Linguistics (ROCLING)*, pp. 19–33, 1997.

[93] Alvarez, M. A. and Lim, S. Discovering interchangeable words from string databases. In *International Conference on Digital Information Management (ICDIM)*, pp. 25–30, Lyon, France, 2007.

[94] Park, E., Cavazos, J., and Alvarez, M. A. Using graph-based program characterization for predictive modeling. In *International Symposium on Code Generation and Optimization (CGO)*, San Jose, CA, USA, 2012, to appear.

Curriculum Vitae

Marco A. Alvarez

++1 (435) 757-4265
Marco.Alvarez@aggiemail.usu.edu

Education

2011 Ph.D. in CS, Utah State University
Graph Kernels and Applications in Bioinformatics
Advisors: Xiaojun Qi and Changhui Yan

1999 M.Sc. in CS, Universidade de São Paulo, Brazil
Pruning Techniques for Artificial Neural Networks
Advisor: Andre Ferreira de Carvalho

1997 B.Sc. in CS, Universidade Federal de Mato Grosso do Sul, Brazil

Publications

Journal Papers and Magazines (refereed)

1. Alvarez, M. A. and C. Yan (2012). A New Protein Graph Model for Function Prediction. *Computational Biology and Chemistry*. to appear.
2. Alvarez, M. A., X. Qi, and C. Yan (2011). A Shortest-Path Graph Kernel for Estimating Gene Product Semantic Similarity. *Journal of Biomedical Semantics* **2**(1), 3.
3. Alvarez, M. A. and C. Yan (2011). A Graph-Based Semantic Similarity Measure for the Gene Ontology. *Journal of Bioinformatics and Computational Biology*.
4. Shelton, B., J. Scoresby, T. Stowell, M. Capell, M. A. Alvarez, and C. Coats (2010). A Frankenstein Approach to Open Source: The Construction of a 3D Game Engine as Meaningful Educational Process. *IEEE Transactions on Learning Technologies* **3** (2), 85–90.
5. Alvarez, M. A., J. Baiocchi, and J. Pow-Sang (2008). Computing and Higher Education in Peru. *Inroads* **40** (2), 35–39.

Conference Papers (refereed)

1. Park, E., J. Cavazos, and M. A. Alvarez (2012). Using Graph-Based Program Characterization for Predictive Modeling. In: *International Symposium on Code Generation and Optimization (CGO)*. to appear. San Jose, CA, USA.
2. Alvarez, M. A. and C. Yan (2010). Exploring Structural Modeling of Proteins for Kernel-Based Enzyme Discrimination. In: *Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. Montreal, Canada, pp.1–5.
3. Shelton, B., M. A. Alvarez, M. Capell, C. C., J. Scoresby, and T. Stowell (2008). Iterations of an Open-Source 3D Game Engine: Multiplayer Environments for Learners. In: *Meaningful Play*. East Lansing, MI, USA.
4. Alvarez, M. A. and S. Lim (2007). A Graph Modeling of Semantic Similarity between Words. In: *International Conference on Semantic Computing (ICSC)*. Irvine, CA, USA, pp.355–362.

5. Alvarez, M. A. and S. Lim (2007). Discovering Interchangeable Words from String Databases. In: *International Conference on Digital Information Management (ICDIM)*. Lyon, France, pp.25–30.
6. Rodrigues, R., R. Viana, A. Pasquali, H. Pistori, and M. A. Alvarez (2007). Máquinas de Vetores de Suporte Aplicadas à Classificação de Defeitos em Couro Bovino. In: *Workshop de Visão Computacional (WVC)*. São José do Rio Preto, SP, Brasil.
7. Viana, R., R. Rodrigues, M. A. Alvarez, and H. Pistori (2007). SVM with Stochastic Parameter Selection for Bovine Leather Defect Classification. In: *Pacific Rim Conference on Advances in Image and Video Technology (PSIVT)*. Santiago, Chile, pp.600–612.
8. Cuadros, E., M. A. Alvarez, and A. de Carvalho (1998). A Multi-Threaded Object Oriented Simulator for Ontogenic Neural Networks. In: *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA)*. Gippsland, Australia, pp.276–281.
9. Cuadros, E., M. A. Alvarez, and A. de Carvalho (1997). Kipu: Um Simulador de Redes Neurais Ontogênicas Orientado a Objetos. In: *Simpósio Brasileiro de Redes Neurais (SBRN)*. Goiânia, GO, Brasil.

Book Chapters

1. Alvarez, M. A., X. Qi, and C. Yan (2011). "GO-Based Term Semantic Similarity". In: *Ontology Learning and Knowledge Discovery Using the Web: Challenges and Recent Advances*. IGI Publishing. Chap. IX.
2. Alvarez, M. A. and S. Lim (2008). "A Machine Learning Approach for One-Stop Learning". In: *Data Mining and Knowledge Discovery Technologies*. IGI Publishing. Chap. XIV.

Theses

1. Alvarez, M. A. (2011). "Graph Kernels and Applications in Bioinformatics". Committee: Adele Cutler, Changhui Yan, Minghui Jiang, Vicki Allan, and Xiaojun Qi. PhD Dissertation. Department of Computer Science, Utah State University.
2. Alvarez, M. A. (1999). "Um Estudo Comparativo de Técnicas de Pruning para Redes Neurais Artificiais". Thesis Committee: André de Carvalho, Heloisa Camargo, and Maria Carolina Monard. Dissertação de Mestrado. Instituto de Ciencias Matemáticas e de Computação, Universidade de São Paulo.

Unpublished Work

1. Alvarez, M. A. (2005). *Modernización de la Infraestructura de Software, Hardware y Comunicaciones en los Sistemas de Información de la Municipalidad Provincial de Tacna*. Proyecto SNIP 20494 (US$ 579.852), Ministerio de Economía y Finanzas, Lima, Perú.
2. Cuadros, E., M. A. Alvarez, and A. de Carvalho (1997). *Simulador Multi-threads de Redes Neurais Ontogênicas Orientado a Objetos*. Technical Report 59. ICMC-USP.

Experience

Positions

Fall 2011 — Research Assistant, Electrical and Computer Engineering Department, Utah State University

I am currently working on a project for automatic authorship attribution in text documents. Preliminary results show improvements over state-of-the-art. Supervisor: Todd Moon

09/10 – 08/11 — Research Assistant, Department of Computer Science, Utah State University

I worked on the representation of gene products as graphs by mining their annotating terms in the Gene Ontology. Then, I proposed the use of graph kernels for estimating functional similarity between pairs of gene products. Supervisors: Xiaojun Qi and Changhui Yan

Summer 2010 Software Engineer Intern, Google Inc., Santa Monica, CA

I worked on the visualization of features extracted from OCR images. In addition, I developed a hierarchical structure of features with the goal of improving the performance of character recognition classifiers. Supervisors: John Flynn and Harmut Neven

09/08 – 05/10 Research Assistant, Department of Computer Science, Utah State University

I proposed different graph representations for proteins based on the 3D coordinates of their respective atoms. By using graph kernels and support vector machines we improved the accuracy for predicting proteins as enzymes. In parallel, I designed a new method for measuring the semantic similarity between pairs of Gene Ontology terms. Supervisors: Changhui Yan and Xiaojun Qi

Summer 2008 Instructor, Department of Computer Science, Utah State University

CS1410 (Introduction to Computer Science – CS2)

09/07 – 08/08 Graduate Assistant, Department of Instructional Technology, Utah State University

I was part of the development team of a 3D game engine for training emergency response personnel. I was responsible for programming the interaction between fire and water particles. Additionally, I created a module for the engine that will allow the use of sound effects in the game. Supervisor: Brett Shelton

01/06 – 08/07 Research Assistant, Department of Computer Science, Utah State University

I proposed a novel semantic similarity measure for pairs of English words. Supervisor: Seungjin Lim

03/05 – 08/06 Vice President, Peruvian Computer Society

I was involved with promoting the Computing field within government agencies and higher education institutions

Fall 2005 Invited Professor, Master in Informatics, Universidad Nacional del Altiplano, Peru

Algorithms, Introduction to Logic

Spring 2005 Invited Professor, Master in Systems Engineering and Informatics, Universidad Nacional Jorge Basadre Grohmann, Peru

Computer Networks and Telecommunications

01/05 – 12/05 Director, Globaltech Solutions, Tacna, Peru

02/02 – 12/04 Academic Coordinator, Graduate Program (Lato Sensu) in Computer Networks, Universidade Católica Dom Bosco, Brazil

02/02 – 12/04 Professor, Graduate Program (Lato Sensu) in Computer Networks, Universidade Católica Dom Bosco, Brazil

Introduction to Computer Networks, Internet and TCP/IP Architecture

01/01 – 12/04 Head of the Program, Engenharia de Computação, Universidade Católica Dom Bosco, Brazil

03/99 – 12/04 Professor, Engenharia de Computação, Universidade Católica Dom Bosco, Brazil

Analysis of Algorithms, Algorithms and Programming, Data Structures 1, Data Structures 2, Computer Networks, Distributed Systems, Artificial Neural Networks, Artificial Intelligence, Formal Languages and Automata Theory, Signal Processing, Computer Graphics

03/99 – 12/00 Technical Coordinator, Informatics Laboratory, Universidade Católica Dom Bosco, Brazil

02/98 – 03/99 Invited Professor, Department of Computer Science, Universidade de Franca, Brazil

Operating Systems, Theory of Computation

01/96 – 03/97 System Analyst, Dataprev – INSS, Campo Grande, MS, Brazil

Honours and Awards

2011	Scholarship to attend the International Conference on Machine Learning (ICML), Bellevue, WA, USA
2011	Scholarship to attend the Machine Learning Summer School (MLSS) at Purdue University, West Lafayette, IN, USA
2011	Travel grant for the Richard Tapia Celebration of Diversity in Computing Conference, San Francisco, CA, USA
2011	Travel grant for the NSF-Sponsored Academic Workshop for Underrepresented Assistant & Associate Professors, and Senior Doctoral Students, Los Angeles, CA, USA
2010	1st place in the Computer Science session of the 13th Annual Intermountain Graduate Research Symposium (cash prize), Utah State University, Logan, UT, USA
2010	Scholarship to attend the 2010 Google GRAD CS Forum, Mountain View, CA, USA
2008	Graduate Student Senate Enhancement Award (scholarship), Utah State University, Logan, UT, USA
2004	Huésped Distinguido (special honor in recognition of his contribution to academic teaching), Rectoral Resolution 3962-2004-R-UPAO, Universidad Privada Antenor Orrego, Trujillo, Peru
2001-2004	Member of the University Council at Universidade Católica Dom Bosco, Campo Grande, MS, Brazil
1997-1999	Scholarship for Master studies at the Universidade de São Paulo. Granted by CAPES – Brazilian Ministry of Education

Invited Talks and Seminars

1. *Graph Kernels in Machine Learning* (2011). Nov 23rd, I Congreso Internacional de Sistemas y Ciencias de la Computación, Universidad Nacional de Moquegua. Ilo, Peru.
2. *Minería de Datos y Reconocimiento de Patrones* (2011). Nov 5-14th, Grupo MARA. Tacna, Peru.
3. *Graph Kernels in Bioinformatics* (2010). Oct 21st, II COREISC - Congreso Regional de Estudiantes de Ingeniería de Sistemas y Computación, Universidad Nacional Jorge Basadre Grohmann. Tacna, Peru.
4. *Graph Kernels and Enzyme Discrimination* (2010). Mar 31st, 13th Annual Intermountain Graduate Research Symposium, Utah State University. Logan, UT, USA.
5. *Graph Kernels and Enzyme Discrimination* (2009). Nov 25th, VIII CSPC - Congreso de la Sociedad Peruana de Computación, Universidad Continental. Huancayo, Peru.
6. *Multicore Programming* (2009). Nov 24-25th, VIII CSPC - Congreso de la Sociedad Peruana de Computación, Universidad Continental. Huancayo, Peru.
7. *Educación Superior en Computación* (2009). Nov 20th, I COMTEL - Congreso Internacional de Computación y Telecomunicaciones, Universidad Inca Garcilaso de la Vega. Lima, Peru.
8. *Educación Superior en Computación* (2009). Nov 20th, Universidad Nacional de Ingeniería. Lima, Peru.
9. *Incubadoras de Empresas Tecnológicas y su Impacto en el Desarrollo Local* (2009). Aug 27th, I COINCO - Congreso Internacional de Ingeniería Comercial, Universidad Nacional Jorge Basadre Grohmann. Tacna, Peru.
10. *SVMs and Kernel Methods* (2008). Nov 13th, VII JPC - Jornada Peruana de Computación, Universidad Nacional Mayor de San Marcos. Lima, Peru.

11. *Computación: Porqué es Importante para el Desarrollo Nacional* (2008). Jun 16th, Universidad Inca Garcilaso de la Vega. Lima, Peru.
12. *Tendencias y Áreas de Investigación en Ciencia de la Computación* (2008). Jun 16th, Universidad Nacional de Ingeniería. Lima, Peru.
13. *Perspectivas de Desarrollo de la Computación e Informática en el País* (2008). Jun 13th, Universidad Nacional Mayor de San Marcos. Lima, Peru.
14. *A Frankenstein Approach to Open Source: The Construction of a 3D Game Engine as a Meaningful Educational Process* (2008). Jun 13th, IV WECI - Workshop Peruano de Educación Superior en Computación e Informática, Pontificia Universidad Católica del Perú. Lima, Peru.
15. *Educación Superior en Computación en el Perú* (2008). Jun 12th, IV WECI - Workshop Peruano de Educación Superior en Computación e Informática, Pontificia Universidad Católica del Perú. Lima, Peru.
16. *A Graph Modeling of Semantic Similarity Between Words* (2008). Apr 2nd, 11th Annual Intermountain Graduate Student Research Symposium, Utah State University. Logan, UT, USA.
17. *Computación: Porqué es Importante para el Desarrollo Nacional* (2007). Aug 9th, II CONEPISIC - Conferencia Nacional de Directores de Escuelas, Carreras y Programas Profesionales de Ingeniería de Sistemas, Informática y Computación, Universidad Privada del Norte. Trujillo, Peru.
18. *Automatic Image Annotation* (2007). Aug 9th, XV CONEIS - Congreso Nacional de Estudiantes de Ingeniería de Sistemas, Universidad Privada del Norte. Trujillo, Peru.
19. *Computación: Porqué es Importante para el Desarrollo Nacional* (2007). Aug 5th, II SIC - Simposio Internacional de las Ciencias - Ciencia y Tecnología para el Desarrollo del Perú, Sheraton Hotel. Lima, Peru.
20. *Measuring the Semantic Similarity of Words* (2006). Nov 27th, V COISIS - Congreso de Informática y Sistemas de Suramérica, Universidad Nacional Jorge Basadre Grohmann. Tacna, Peru.
21. *Una Breve Introducción a la Minería de Datos* (2006). Oct 5th, I Jornadas Académicas de Ingeniería Informática, Universidad Juan Misael Saracho. Tarija, Bolivia.
22. *Minería de Datos: Métodos, Técnicas y Aplicaciones* (2006). Jun 28th, V JPC - Jornada Peruana de Computación. Arequipa, Peru.
23. *Tutorial de Minería de Datos* (2006). Jun 26th, Universidad Nacional Jorge Basadre Grohmann. Tacna, Peru.
24. *Minería de Datos: Métodos, Técnicas y Aplicaciones* (2006). Jun 26th, Colegio de Ingenieros del Perú. Tacna, Peru.
25. *Tutorial de Minería de Datos* (2006). Jun 22-23rd, Pontificia Universidad Católica del Perú. Lima, Peru.
26. *Maratón de Programación - ACM ICPC* (2005). Nov 29th, VI Workshop SIIS, Universidad Nacional Jorge Basadre Grohmann. Tacna, Peru.
27. *Computación e Informática: Perspectivas de la Educación Superior* (2005). Nov 28th, VI Workshop SIIS, Universidad Nacional Jorge Basadre Grohmann. Tacna, Peru.
28. *Computación e Informática: Perspectivas de la Educación Superior* (2005). Nov 22nd, IV COISIS - Congreso de Informática y Sistemas de Suramérica, Universidad Nacional del Altiplano. Puno, Peru.
29. *Computación e Informática: Perspectivas de la Educación Superior* (2005). Nov 16th, II CICIS - Congreso Internacional de Computación, Informática y Sistemas, Universidad José Carlos Mariátegui. Moquegua, Peru.
30. *Incubadoras de Empresas* (2005). Nov 9th, II Congreso Internacional de Ingeniería de Computación y Sistemas, Universidad Privada Antenor Orrego. Trujillo, Peru.
31. *Computación e Informática: Perspectivas de la Educación Superior* (2005). Nov 7th, VI Seminario sobre Diseño de Currículo, Asamblea Nacional de Rectores. Lima, Peru.

32. *Perfiles Profesionales en el área de Computación e Informática* (2005). Oct 22nd, II Seminario Internacional en Ciencia de la Computación, Universidad Nacional del Altiplano. Puno, Peru.
33. *Perfiles Profesionales en el área de Computación e Informática* (2005). Oct 20th, I CONIIC - Congreso sobre Ingeniería e Investigación Científica, Universidad Tecnológica del Perú. Lima, Peru.
34. *Redes Neuronales Artificiales: Teoría y Aplicaciones* (2005). Oct 19th, I CONIIC - Congreso sobre Ingeniería e Investigación Científica, Universidad Tecnológica del Perú. Lima, Peru.
35. *Software Libre y la Producción Cooperativa de Conocimiento* (2005). Oct 13th, VI Encuentro PostMaster, Universidad Nacional Jorge Basadre Grohmann. Tacna, Peru.
36. *Métodos y Aplicaciones de Sistemas Inteligentes* (2005). Oct 13th, VI Encuentro PostMaster, Universidad Nacional Jorge Basadre Grohmann. Tacna, Peru.
37. *Perfiles Profesionales en el área de Computación e Informática* (2005). Sep 29th, III WECI - Workshop Peruano de Educación Superior en Computación e Informática, Colegio de Ingenierios del Peru. Lima, Peru.
38. *El Impacto del Avance Tecnológico en la Sociedad Contemporanea* (2005). Aug 13th, X Congreso International de Secretarias y Asistentes Ejecutivas. Arica, Chile.
39. *Perfiles Profesionales en el área de Computación e Informática* (2005). Jul 17th, III Congreso Internacional en Ingeniería de Sistemas, Universidad Privada Antenor Orrego. Trujillo, Peru.
40. *Redes Neuronales Artificiales: Teoría y Aplicaciones* (2005). Jul 14th, III Congreso Internacional en Ingeniería de Sistemas, Universidad Privada Antenor Orrego. Trujillo, Peru.
41. *Redes Neuronales Artificiales: Teoría y Aplicaciones* (2005). May 28th, IV JPC - Jornada Peruana de Computación e Informática. Tacna, Peru.
42. *Perfiles Profesionales en el área de Computación e Informática* (2005). May 27th, IV JPC - Jornada Peruana de Computación e Informática. Tacna, Peru.
43. *Aspectos y Tendencias Globales en Tecnología de la Información* (2005). Apr 23rd, Encuentro de Asistentes de Gerencia. Moquegua, Peru.
44. *Workflow: Trascendiendo la Automatización Organizacional* (2005). Apr 2nd, Encuentro de Asistentes de Gerencia, Camino Real Hotel. Tacna, Peru.
45. *La Opción del Código Abierto* (2005). Mar 14th, ZOFRATACNA. Tacna, Peru.
46. *Incubadoras de Empresas* (2005). Mar 9th, ZOFRATACNA. Tacna, Peru.
47. *La Tecnología y la Producción Cooperativa de Conocimiento: Reflexiones e Impactos en el Sector Educativo* (2005). Jan 13th, I Congreso Internacional de Informática Educativa. Tacna, Peru.
48. *La Computación: Reflexiones del área y contribuciones para su desarrollo en la región* (2005). Jan 7th, Colegio de Ingenieros del Perú. Tacna, Peru.
49. *La Computación: Reflexiones en el área y propuestas para la educación superior en el Perú* (2004). Nov 13th, II WECI - Workshop Peruano de Educación Superior en Computación e Informática, Universidad Privada Antenor Orrego. Trujillo, Peru.
50. *Maratón de Programación - ACM ICPC* (2004). Nov 9th, Universidad Privada Antenor Orrego. Trujillo, Peru.
51. *Maratona de Programação - ACM ICPC* (2004). Aug 31st, UNIDERP. Campo Grande, MS, Brazil.
52. *Maratona de Programação - ACM ICPC* (2004). Aug 25th, UNAES. Campo Grande, MS, Brazil.
53. *Pesquisa em Computação na Universidade Católica Dom Bosco* (2004). May 29th, II Seminário de Pesquisa em Ciência da Computação, UNAES. Campo Grande, MS, Brazil.
54. *Aspectos e Tendências Globais em Tecnologia da Informação* (2004). Mar 25th, UNIDERP. Rio Verde, MS, Brazil.
55. *O Grupo de Pesquisa em Engenharia e Computação na Universidade Católica Dom Bosco* (2003). May 17th, I Seminário de Pesquisa em Ciência da Computação, UNAES. Campo Grande, MS, Brazil.

56. *Redes Neuronales Artificiales: Teoría y Práctica* (2003). Jan 18th, Pontificia Universidad Católica del Perú. Lima, Peru.
57. *Análisis y Tendencias de la Computación en el Plano Educacional* (2003). Jan 2nd, I Congreso Internacional de Científicos Peruanos, Marriot Hotel. Lima, Peru.
58. *Intel IA-32 Architecture* (2001). Aug 30th, Universidade Católica Dom Bosco. Campo Grande, MS, Brazil.
59. *Objetos Distribuídos y OMG CORBA* (2001). Aug 27th, Universidad Columbia del Paraguay. Pedro Juan Caballero, Paraguay.

Supervision of Undergraduate Final Projects

1. Daniel Santos Silva and Narumi Abe (2002). *Redes Neurais Artificiais aplicadas ao Reconhecimento de Impressões Digitais*. Engenharia de Computação, Universidade Católica Dom Bosco. Brazil.
2. Alexandre Rapchan and Raphael Maia Valente (2001). *Construção de um Portal na Web com Serviços e Informações da Cidade de Campo Grande*. Engenharia de Computação, Universidade Católica Dom Bosco. Brazil.
3. Bianca Roriz de Carvalho and Cynara Liz Saad (2001). *Utilização de RNAs para Classificação de Padrões Bioclimáticos a partir de Dados NDVI FFT*. Engenharia de Computação, Universidade Católica Dom Bosco. Brazil.
4. Carlos Alberto Vasconcelos and Thiago Cardoso (2001). *Implementação de um Módulo Localizador de Placas em Imagens de Veículos*. Engenharia de Computação, Universidade Católica Dom Bosco. Brazil.
5. Fatten Ramani Biacio and Fabiana Rocha da Lima (2001). *Protótipo de um Reconhecedor Automático de Palavras Isoladas Independente do Locutor Utilizando Redes Neurais Artificiais*. Engenharia de Computação, Universidade Católica Dom Bosco. Brazil.
6. Cristiano Dias and Wallace Lemos (2000). *Aplicação de Técnicas de Data Mining para a Descoberta de Conhecimento no Acesso a Servidores WEB*. Engenharia de Computação, Universidade Católica Dom Bosco. Brazil.
7. Cristiano Pires Martins and Arilson Silva de Oliveira (2000). *Utilização de Redes Neurais Artificiais para Detecção de Intrusos*. Engenharia de Computação, Universidade Católica Dom Bosco. Brazil.

Languages

Spanish	Fluent
Portuguese	Fluent
English	Fluent